

## Test Solutions - Programming Manual

# Modular Test Systems



**ZTM Series** Rack-Mount Modular Test Systems  
**RCM Series** Compact Modular Test Systems



## **Important Notice**

This guide is owned by Mini-Circuits and is protected by copyright, trademark and other intellectual property laws.

The information in this guide is provided by Mini-Circuits as an accommodation to our customers and may be used only to promote and accompany the purchase of Mini-Circuits' Parts. This guide may not be reproduced, modified, distributed, published, stored in an electronic database, or transmitted and the information contained herein may not be exploited in any form or by any means, electronic, mechanical recording or otherwise, without prior written permission from Mini-Circuits.

This guide is subject to change, qualifications, variations, adjustments or modifications without notice and may contain errors, omissions, inaccuracies, mistakes or deficiencies. Mini-Circuits assumes no responsibility for, and will have no liability on account of, any of the foregoing. Accordingly, this guide should be used as a guideline only.

## **Trademarks**

Microsoft, Windows, Visual Basic, Visual C# and Visual C++ are registered trademarks of Microsoft Corporation. LabVIEW and CVI are registered trademarks of National Instruments Corporation. Delphi is a registered trademark of Delphi Technologies, Inc. MATLAB is a registered trademark of The MathWorks, Inc. Agilent VEE is a registered trademark of Agilent Technologies, Inc. Linux is a registered trademark of Linus Torvalds. Mac is a registered trademark of Apple Inc. Python is a registered trademark of Python Software Foundation Corporation.

All other trademarks cited within this guide are the property of their respective owners. Neither Mini-Circuits nor the Mini-Circuits PTE (portable test equipment) series are affiliated with or endorsed or sponsored by the owners of the above referenced trademarks.

Mini-Circuits and the Mini-Circuits logo are registered trademarks of Scientific Components Corporation.

## **Mini-Circuits**

13 Neptune Avenue

Brooklyn, NY 11235, USA

Phone: +1-718-934-4500

Email: [sales@minicircuits.com](mailto:sales@minicircuits.com)

Web: [www.minicircuits.com](http://www.minicircuits.com)

<b>1 - Overview.....</b>	<b>6</b>
<b>2 - Programming with Mini-Circuits' Modular Test Systems.....</b>	<b>7</b>
2.1 - Control Options.....	7
2.2 - Addressing Individual Test Components.....	7
2.3 - Example ZTM Series Configuration.....	8
2.4 - Example RCM-100 Series Configuration .....	9
2.5 - Example RCM-200 Series Configuration .....	10
<b>3 - SCPI Commands for Control of Modular Test Components .....</b>	<b>11</b>
<b>3.1 - ZTM Series System Operations.....</b>	<b>12</b>
3.1 (a) - Get Model Name .....	13
3.1 (b) - Get Serial Number.....	14
3.1 (c) - Get Configuration .....	15
3.1 (d) - Get Firmware .....	17
3.1 (e) - Get Internal Temperature .....	18
3.1 (f) - Get Heat Alarm .....	19
3.1 (g) - Save Counters & States .....	20
<b>3.2 - Programmable Attenuator Control .....</b>	<b>21</b>
3.2 (a) - Set Attenuation .....	22
3.2 (b) - Get Attenuation .....	23
3.2 (c) - Set Start-Up Attenuation Mode .....	24
3.2 (d) - Get Start-Up Attenuation Mode .....	25
3.2 (e) - Set Start-Up Attenuation Value.....	26
3.2 (f) - Get Start-Up Attenuation Value.....	27
3.2 (g) - Get Maximum Attenuation .....	28
<b>3.3 - SPDT Switch Control .....</b>	<b>29</b>
3.3 (a) - Set SPDT Switch State.....	30
3.3 (b) - Get SPDT Switch State.....	31
3.3 (c) - Set All SPDT Switch States .....	32
3.3 (d) - Get All SPDT Switch States .....	34
<b>3.4 - SP4T Switch Control.....</b>	<b>36</b>
3.4 (a) - Set SP4T Switch State .....	37
3.4 (b) - Get SP4T Switch State .....	38
3.4 (c) - Set All SP4T Switch States.....	39
3.4 (d) - Get All SP4T Switch States.....	41
<b>3.5 - SP6T Switch Control.....</b>	<b>43</b>
3.5 (a) - Set SP6T Switch State .....	44
3.5 (b) - Get SP6T Switch State .....	45
3.5 (c) - Set All SP6T Switch States.....	46
3.5 (d) - Get All SP6T Switch States.....	48
<b>3.6 - SP8T Switch Control.....</b>	<b>50</b>
3.6 (a) - Set SP8T Switch State .....	51
3.6 (b) - Get SP8T Switch State .....	52
3.6 (c) - Set All SP8T Switch States.....	53
3.6 (d) - Get All SP8T Switch States.....	55
<b>3.7 - Transfer Switch Control .....</b>	<b>57</b>
3.7 (a) - Set Transfer Switch State .....	58
3.7 (b) - Get Transfer Switch State.....	59
3.7 (c) - Set All Transfer Switch States .....	60
3.7 (d) - Get All Transfer Switch States .....	62

<b>3.8 - Switch Start-Up and Counter Properties.....</b>	<b>64</b>
3.8 (a) - Set Switch Start-Up Mode .....	65
3.8 (b) - Get Switch Start-Up Mode .....	66
3.8 (c) - Get Switch Counter.....	67
<b>3.9 - Component Labels.....</b>	<b>69</b>
3.9 (a) - Set Component Label .....	69
3.9 (b) - Get Component Label .....	70
<b>3.10 - SCPI - Ethernet Configuration Commands.....</b>	<b>71</b>
3.10 (a) - Set Static IP Address .....	72
3.10 (b) - Get Static IP Address .....	73
3.10 (c) - Set Static Subnet Mask .....	74
3.10 (d) - Get Static Subnet Mask .....	75
3.10 (e) - Set Static Network Gateway.....	76
3.10 (f) - Get Static Network Gateway.....	77
3.10 (g) - Set HTTP Port.....	78
3.10 (h) - Get HTTP Port.....	79
3.10 (i) - Set Telnet Port.....	80
3.10 (j) - Get Telnet Port .....	81
3.10 (k) - Set Password Requirement .....	82
3.10 (l) - Get Password Requirement .....	83
3.10 (m) - Set Password .....	84
3.10 (n) - Get Password .....	85
3.10 (o) - Set DHCP Status .....	86
3.10 (p) - Get DHCP Status.....	87
3.10 (q) - Get MAC Address .....	88
3.10 (r) - Get Current Ethernet Configuration .....	89
3.10 (s) - Update Ethernet Settings .....	90
<b>4 - Operating in a Windows Environment via USB .....</b>	<b>91</b>
<b>4.1 - The DLL (Dynamic Link Library) Concept.....</b>	<b>91</b>
4.1 (a) - ActiveX COM Object .....	92
4.1 (b) - Microsoft.NET Class Library .....	94
<b>4.2 - Referencing the DLL (Dynamic Linked Library).....</b>	<b>95</b>
<b>4.3 - Summary of DLL Functions.....</b>	<b>96</b>
4.3 (a) - USB Control Functions.....	96
4.3 (b) - Ethernet Configuration Functions.....	96
<b>4.4 - DLL Functions for USB Control.....</b>	<b>97</b>
4.4 (a) - Connect by Serial Number.....	97
4.4 (b) - Connect by Address.....	98
4.4 (c) - Disconnect .....	99
4.4 (d) - Read Model Name.....	100
4.4 (e) - Read Serial Number.....	101
4.4 (f) - Set USB Address .....	102
4.4 (g) - Get USB Address.....	103
4.4 (h) - Get List of Connected Serial Numbers .....	104
4.4 (i) - Get List of Available Addresses .....	105
4.4 (j) - Get Software Connection Status .....	106
4.4 (k) - Get USB Connection Status .....	107
4.4 (l) - Send SCPI Command .....	108
4.4 (m) - Get Firmware .....	109
4.4 (n) - Get Firmware Version (Antiquated).....	110

<b>4.5 - DLL Functions for Ethernet Configuration.....</b>	<b>111</b>
4.5 (a) - Get Ethernet Configuration.....	111
4.5 (b) - Get IP Address.....	113
4.5 (c) - Get MAC Address.....	115
4.5 (d) - Get Network Gateway.....	117
4.5 (e) - Get Subnet Mask.....	119
4.5 (f) - Get TCP/IP Port.....	121
4.5 (g) - Get DHCP Status.....	122
4.5 (h) - Get Password Status.....	123
4.5 (i) - Get Password.....	124
4.5 (j) - Save IP Address.....	125
4.5 (k) - Save Network Gateway.....	126
4.5 (l) - Save Subnet Mask.....	127
4.5 (m) - Save TCP/IP Port.....	128
4.5 (n) - Use DHCP.....	129
4.5 (o) - Use Password.....	130
4.5 (p) - Set Password.....	131
4.5 (q) - Set Telnet Prompt.....	132
4.5 (r) - Get Telnet Prompt Status.....	133
<b>5 - Operating in a Linux Environment via USB.....</b>	<b>134</b>
<b>5.1 - Summary of Commands.....</b>	<b>134</b>
<b>5.2 - Detailed Description of Commands.....</b>	<b>135</b>
5.2 (a) - Get Device Model Name.....	135
5.2 (b) - Get Device Serial Number.....	136
5.2 (c) - Send SCPI Command.....	137
5.2 (d) - Get Firmware.....	140
5.2 (e) - Get Internal Temperature.....	141
<b>6 - Ethernet Control over IP Networks.....</b>	<b>143</b>
<b>6.1 - Ethernet Communication.....</b>	<b>144</b>
6.1 (a) - Sending SCPI Commands/Queries Using HTTP.....	144
6.1 (b) - Sending SCPI/Commands/Queries Using Telnet.....	145
6.1 (c) - Device Discovery Using UDP.....	146
<b>7 - Program Examples &amp; Tutorials.....</b>	<b>148</b>
<b>7.1 - Perl Programming.....</b>	<b>148</b>
7.1 (a) - Ethernet HTTP Connection Using Perl's LWP Simple Interface.....	148
7.1 (b) - USB Connection Using the ActiveX DLL in 32-bit Perl Distributions.....	148
7.1 (c) - Work-Around for 64-bit Perl Distributions Using USB Connection.....	149
<b>7.2 - C# Programming.....</b>	<b>150</b>
7.2 (a) - Creating an Executable Using the .Net DLL in C# for USB Control.....	150
<b>7.3 - LabVIEW.....</b>	<b>151</b>
7.3 (a) - Creating a LabVIEW VI for USB Control with the ActiveX DLL.....	151

## 1 - Overview

This Programming Manual is intended for customers wishing to create their own interface for Mini-Circuits' ZTM & RCM Series modular test systems. For instructions on using the supplied GUI program, or connecting the hardware, please see the User Guide at:

[https://www.minicircuits.com/softwaredownload/ztm\\_rcm.html](https://www.minicircuits.com/softwaredownload/ztm_rcm.html)

Mini-Circuits offers support over a variety of operating systems, programming environments and third party applications.

Support for Windows® operating systems is provided through the Microsoft® .NET® and ActiveX® frameworks to allow the user to develop customized control applications. Support for Linux® operating systems is accomplished using the standard libhid and libusb libraries.

Mini-Circuits has experience with a wide variety of environments including (but not limited to):

- Visual Basic®, Visual C#®, Visual C++®
- Delphi®
- Borland C++®
- CVI®
- LabVIEW®
- MATLAB®
- Python®
- Agilent VEE®

The software package includes a GUI program, ActiveX and .NET DLL files, Linux support, project examples for third party software, and detailed user manuals. The latest package is available for download at:

[https://www.minicircuits.com/softwaredownload/ztm\\_rcm.html](https://www.minicircuits.com/softwaredownload/ztm_rcm.html)

Files made available for download from the Mini-Circuits website are subject to Mini-Circuits' terms of use which are available on the website.

## 2 - Programming with Mini-Circuits' Modular Test Systems

The ZTM & RCM Series modular concept is a flexible test system that can be easily configured to a user's requirements with combinations of high reliability mechanical switches and programmable attenuators.

### 2.1 - Control Options

Communication with the system can be accomplished in a number of ways:

1. Using the provided ActiveX or .Net API objects (DLL files) on a Windows operating system (see [Operating in a Windows Environment via USB](#))
2. Using the USB interrupt codes on a Linux operating system (see [Operating in a Linux Environment via USB](#))
3. Using HTTP or Telnet communication over an Ethernet connection (see [Ethernet Control over IP Networks](#)), which is largely independent of the operating system

In all cases the full functionality of the system and the constituent test components is accessible using a command set based on SCPI (see [SCPI Commands for Control of Modular Test Components](#)).

### 2.2 - Addressing Individual Test Components

The system is logically arranged with a series of component windows on the front panel, 6 for the ZTM Series (labelled 1 to 6) and 3 for the RCM Series (labelled 1 to 3). Each window can be specified with the following components:

- 1 or 2 high reliability, mechanical SPDT switches (DC-18GHz) in positions A and B
- 1 or 2 high reliability, mechanical transfer switches (DC-18GHz) in positions A and B
- 1 high reliability, mechanical SP4T switch (DC-18GHz)
- 1 high reliability, mechanical SP6T switch (DC-12GHz)
- 1 or 2 programmable attenuators:
  - 0 to 120dB (0.25dB step size) covering 1MHz to 4GHz
  - 0 to 30, 60, 90 or 110dB (0.25dB step size) covering 1MHz to 6GHz
  - 0 to 30, 60 or 90dB (0.25dB step size) covering 1MHz to 8GHz

A special case exists for ZTM and RCM Series models housing SP8T switch components. The SP8T components are larger than the standard window size so cannot be mixed with other components. The ZTM Series can be specified with up to 4 SP8T switches and the RCM Series with up to 2 SP8T switches.

Component	Designator	Number per Window (Windows 1 to 6)
SPDT Switch	SPDT	1 or 2 (in locations A and B)
SP4T Switch	SP4T	1 only
SP6T Switch	SP6T	1 only
SP8T Switch	SP8T	1 only (windows 1 to 4 only)
Transfer Switch	MTS	1 or 2 (in locations A and B)
Programmable Attenuator	RUDAT	1 or 2 (in locations A and B)

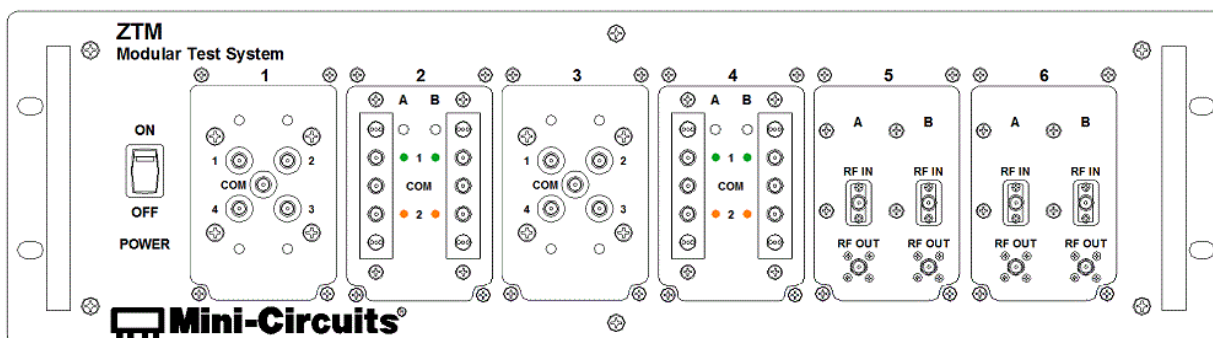
*Table 1: ZTM / RCM Series Component Designators*



#### Notes:

1. A full window is required for SP4T, SP6T and SP8T switches so the “A” and “B” sub-locations do not apply; only the integer window number is needed
2. Transfer switches and SPDT switches accept the same command arguments so an “SPDT” command addressed to an “MTS” location will still operate the switch (and vice versa)

## 2.3 - Example ZTM Series Configuration



**Fig 1: Example ZTM Series Front Panel Configuration**

The example front panel layout of figure 1 is arranged with two SP4T switches, four SPDT switches and four programmable attenuators in the following locations:

ZTM Series Window	Component
1	SP4T Switch
2A	SPDT Switch
2B	SPDT Switch
3	SP4T Switch
4A	SPDT Switch
4B	SPDT Switch
5A	Programmable Attenuator
5B	Programmable Attenuator
6A	Programmable Attenuator
6B	Programmable Attenuator

**Table 2: Example ZTM Series Configuration with Component Window Locations**



## 2.4 - Example RCM-100 Series Configuration

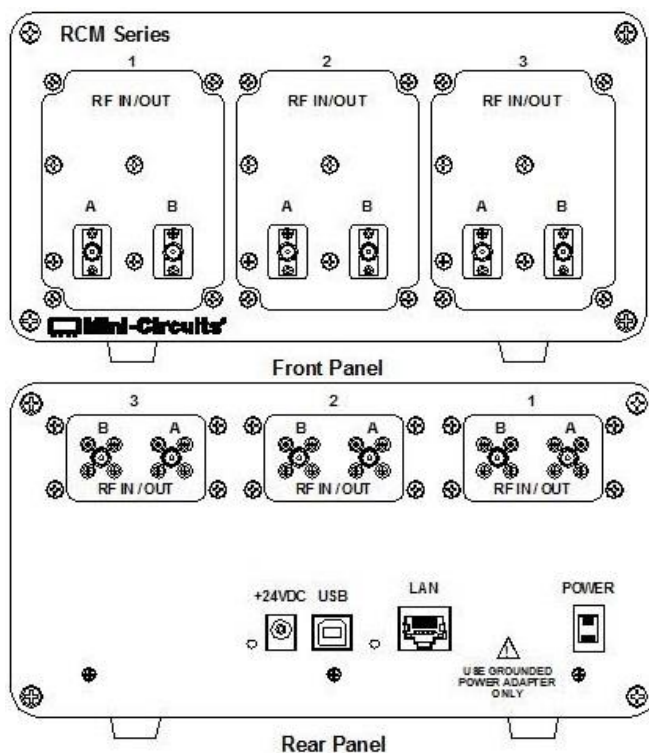


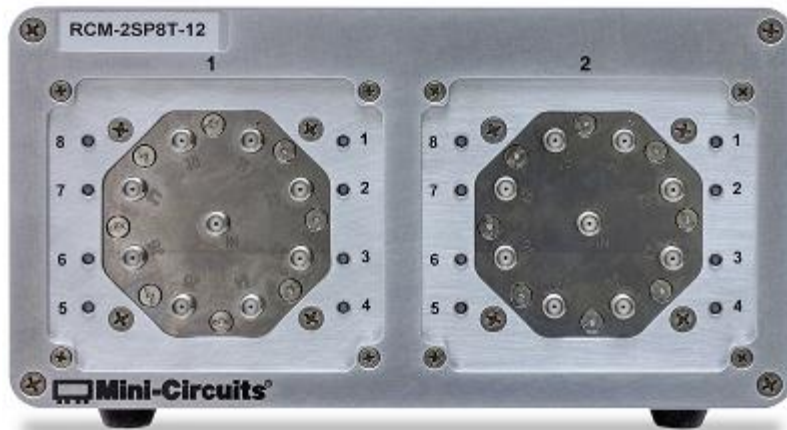
Fig 2: Example RCM-100 Series Configuration

The example front panel layout of figure 2 is arranged with six programmable attenuators in the following locations:

RCM Series Window	Component
1A	6GHz Programmable Attenuator (0 to 30 dB)
1B	6GHz Programmable Attenuator (0 to 30 dB)
2A	6GHz Programmable Attenuator (0 to 60 dB)
2B	6GHz Programmable Attenuator (0 to 60 dB)
3A	6GHz Programmable Attenuator (0 to 110 dB)
3B	6GHz Programmable Attenuator (0 to 110 dB)

Table 3: Example RCM-100 Series Configuration with Component Window Locations

## 2.5 - Example RCM-200 Series Configuration



*Fig 3: Special Case of the RCM-200 Series, Specified with 2 x SP8T Switches (RCM-2SP8T-12)*

The example front panel layout of figure 2 is arranged with six programmable attenuators in the following locations:

RCM Series Window	Component
1	SP8T Switch
2	SP8T Switch

*Table 4: Example RCM-200 Series Configuration with Component Window Locations*

### 3 - SCPI Commands for Control of Modular Test Components

The main method of communication with the ZTM & RCM Series modular test systems is through a series of SCPI commands. SCPI (Standard Commands for Programmable Instruments) is a common method for controlling instrumentation products.

The SCPI commands are sent as an ASCII text string (up to 63 characters) in the below format:

```
: [designator] : [location] : [command] : [value]
```

Where:

- [designator] = the short-name of the component to be controlled (see table 1)
- [location] = the location of the component to be controlled
- [command] = the command/query to send
- [value] = the value (if applicable) to set

Where a SCPI command is sent to query a property of the full modular test system, rather than an individual component, the *Designator* and *Window* parameters are not used (for example [Get Internal Temperature](#)).

Commands can be sent in upper or lower case and the return value will be an ASCII text string.

These commands and queries can be sent using the DLL function [Send SCPI Command](#) when the system is connected via the USB interface in a Microsoft Windows environment, or using the USB interrupt commands on a Linux system (see Linux [Send SCPI Command](#)). In addition, SCPI commands can also be sent using HTTP get/post commands or Telnet over a TCP/IP network when the system is connected via the Ethernet RJ45 port (see [Ethernet Communication](#)).

### 3.1 - ZTM Series System Operations

: [query]

The following queries provide information regarding the modular test system rather than the individual test components, so there is no location or component designator to send with the command.

	Description	Command/Query
a	Get Model Name	MN?
b	Get Serial Number	SN?
c	Get Configuration	CONFIG:APP?
d	Get Firmware	FIRMWARE?
e	Get Internal Temperature	T[sensor]?
f	Get Heat Alarm	HEATALARM?
g	Save Counters & States	OPERATIONDATA:SAVE

### 3.1 (a) - Get Model Name

#### Description

Returns the full Mini-Circuits part number of the connected system.

#### Command Syntax

**:MN?**

#### Return String

**MN=[model]**

Variable	Description
[model]	Full model name of the system (for example, "ZTM-999")

#### Examples

String to Send	String Returned
<b>:MN?</b>	<b>MN=ZTM-999</b>

DLL Implementation: `Send_SCPI(":MN?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:MN?`

#### See Also

[Get Serial Number](#)

### 3.1 (b) - Get Serial Number

#### Description

Returns the serial number of the connected system.

#### Command Syntax

`:SN?`

#### Return String

`SN=[serial]`

Variable	Description
[serial]	Serial number of the system (for example, "11401010001")

#### Examples

String to Send	String Returned
<code>:SN?</code>	<code>SN=11401010001</code>

DLL Implementation: `Send_SCPI(":SN?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:SN?`

#### See Also

[Get Model Name](#)

### 3.1 (c) - Get Configuration

#### Description

Returns a list of integer codes (separated by semi-colons) indicating the component(s) mounted in each window of the system. The possible integer codes are:

- 0 = No component in this window
- 1 = SPDT in sub-location A only
- 2 = SPDT in sub-location B only
- 3 = SPDT in both sub-locations A and B
- 4 = SP4T
- 5 = MTS in sub-location A only
- 6 = MTS in sub-location B only
- 7 = MTS in both sub-locations A and B
- 8 = Attenuator in sub-location A only
- 9 = Attenuator in sub-location B only
- 10 = Attenuator in both sub-locations A and B
- 11 = SP6T
- 12 = SP8T

#### Command Syntax

**:CONFIG:APP?**

#### Return String

**APP=[window1] ; [window2] ; [window3] ; [window4] ; [window5] ; [window6]**

Variable	Description
[window1]	Integer code indicating the component(s) in window 1
[window2]	Integer code indicating the component(s) in window 2
[window3]	Integer code indicating the component(s) in window 3
[window4]	Integer code indicating the component(s) in window 4 Not applicable to RCM Series
[window5]	Integer code indicating the component(s) in window 5 Not applicable to RCM Series
[window6]	Integer code indicating the component(s) in window 6 Not applicable to RCM Series



## Examples

String to Send	String Returned
<code>:CONFIG:APP?</code>	<code>APP=4;3;4;3;10;10</code>
<code>:CONFIG:APP?</code>	<code>APP=10;10;10</code>

“APP=4;3;4;3;10;10” would be returned for the ZTM Series example in figure 1, indicating:

- Window 1 = SP4T switch
- Window 2 = Two SPDT switches
- Window 3 = SP4T switch
- Window 4 = Two SPDT switches
- Window 5 = Two attenuators
- Window 6 = Two attenuators

“APP=10;10;10” would be returned for the RCM Series example in figure 2, indicating:

- Window 1 = Two attenuators
- Window 2 = Two attenuators
- Window 3 = Two attenuators

DLL Implementation: `Send_SCPI(":CONFIG:APP?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:CONFIG:APP?`

### 3.1 (d) - Get Firmware

#### Description

Returns the firmware version of the system.

#### Command Syntax

`:FIRMWARE?`

#### Return String

`FIRMWARE=[firmware]`

Variable	Description
<code>[firmware]</code>	Firmware version name (for example, "A1")

#### Examples

String to Send	String Returned
<code>:FIRMWARE?</code>	<code>FIRMWARE=A1</code>

DLL Implementation: `Send_SCPI(":FIRMWARE?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:FIRMWARE?`

### 3.1 (e) - Get Internal Temperature

#### Description

Returns the internal temperature of the modular test system, measured at the control board.

#### Command Syntax

`:TS0?`

#### Return String

`[temperature]`

Variable	Description
<code>[temperature]</code>	The temperature returned from the specified sensor in degrees Celsius

#### Examples

String to Send	String Returned
<code>:TS0?</code>	25.50

DLL Implementation: `Send_SCPI(":TS0?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:TS0?`

#### See Also

[Get Heat Alarm](#)

### 3.1 (f) - Get Heat Alarm

#### Description

Returns an alarm code relating to the internal temperature of the modular test system.

#### Command Syntax

`:HEATALARM?`

#### Return String

`[alarm]`

Variable	Value	Description
<code>[alarm]</code>	0	Internal temperature within normal limits
	1	Internal temperature has exceeded the factory defined limit (65°C)

#### Examples

String to Send	String Returned
<code>:HEATALARM?</code>	0

DLL Implementation: `Send_SCPI(":HEATALARM?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:HEATALARM?`

#### See Also

[Get Internal Temperature](#)

### 3.1 (g) - Save Counters & States

#### Description

Transfers any operation data from temporary to permanent memory. The data includes:

1. Last switch states
2. Last attenuation values
3. Latest switch counters

This command should be sent following completion of all switch / attenuator sequences and prior to powering off the system in order to preserve the latest data. During normal operation, this data is internally stored in volatile memory but automatically updated into permanent memory every 3 minutes.

#### Requirements

Serial Number Range	Firmware Version
Up to 11412319999	A8 to A99
From 11501010000	B2 or later

#### Command Syntax

**:OPERATIONDATA:SAVE**

#### Return String

**[status]**

Variable	Value	Description
[status]	0 - Failed	Command failed
	1 - Success	Command completed successfully
	2 - Fail	Command already sent within previous 3 minutes (wait and try again)

#### Examples

String to Send	String Returned
:OPERATIONDATA:SAVE	1 - Success

DLL Implementation: `Send_SCPI(":OPERATIONDATA:SAVE", RetStr)`

HTTP Implementation: `http://10.10.10.10/:OPERATIONDATA:SAVE`

#### See Also

[Set Start-Up Attenuation Mode](#)  
[Get Start-Up Attenuation Mode](#)  
[Set Switch Start-Up Mode](#)  
[Get Switch Start-Up Mode](#)  
[Get Switch Counter](#)

## 3.2 - Programmable Attenuator Control

**:RUDAT:** [location] : [command] : [value]

The following commands and queries allow control of a specific programmable attenuator within the modular test system. The component designator “RUDAT” should be used along with the location of the attenuator within the system (for example “1A” or “1B” where there are 2 attenuators in window 1, or just “1” when there is only a single attenuator in window 1).

	Description	Command/Query
<b>a</b>	Set Attenuation	<b>ATT:</b> [value]
<b>b</b>	Get Attenuation	<b>ATT?</b>
<b>c</b>	Set Start-Up Attenuation Mode	<b>STARTUPATT:INDICATOR:</b> [mode]
<b>d</b>	Get Start-Up Attenuation Mode	<b>STARTUPATT:INDICATOR?</b>
<b>e</b>	Set Start-Up Attenuation Value	<b>STARTUPATT:VALUE:</b> [value]
<b>f</b>	Get Start-Up Attenuation Value	<b>STARTUPATT:VALUE?</b>
<b>g</b>	Get Max Attenuation	<b>MAX?</b>

### 3.2 (a) - Set Attenuation

#### Description

Set the attenuation of a specific programmable attenuator.

#### Command Syntax

**:ATT:** [value]

Variable	Description
[value]	The attenuation to set in dB (up to 2 decimal places).

#### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed (attenuation not set)
	1 - Success	Command completed successfully

#### Examples

String to Send	String Returned
:RUDAT:1A:ATT:70.25	1 - Success
:RUDAT:2B:ATT:0	1 - Success

DLL Implementation: `Send_SCPI(":RUDAT:1A:ATT:70.25", RetStr)`

HTTP Implementation: `http://10.10.10.10/:RUDAT:1A:ATT:70.25`

#### See Also

[Get Attenuation](#)

[Get Maximum Attenuation](#)



### 3.2 (b) - Get Attenuation

#### Description

Read the attenuation of a specific programmable attenuator.

#### Command Syntax

`:ATT?`

#### Return String

`[value]`

Variable	Description
<code>[value]</code>	The attenuation setting (in dB) as a string of ASCII characters

#### Examples

String to Send	String Returned
<code>:RUDAT:1A:ATT?</code>	<code>70.25</code>
<code>:RUDAT:2B:ATT?</code>	<code>0.00</code>

DLL Implementation: `Send_SCPI(":RUDAT:1A:ATT?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:RUDAT:1A:ATT?`

#### See Also

[Set Attenuation](#)

[Get Maximum Attenuation](#)

### 3.2 (c) - Set Start-Up Attenuation Mode

#### Description

Sets the start-up mode of a specific attenuator when the system is powered up. The attenuator can be configured to load the last remembered value, a specific fixed value or the maximum possible attenuation.

#### Command Syntax

**:STARTUPATT:INDICATOR:[mode]**

Variable	Value	Description
[mode]	L	Last Value - the attenuator will load the last remembered attenuation See <a href="#">Save Counters &amp; States</a> for correct operation
	F	Fixed Value - the attenuator will load a fixed value (see Set Start-Up Attenuation Value)
	N	Normal - the attenuator will load the maximum possible attenuation (this is the default setting)

#### Return String

**[status]**

Variable	Value	Description
[status]	0 - Failed	Command failed
	1 - Success	Command completed successfully

#### Examples

String to Send	String Returned
:RUDAT:1A:STARTUPATT:INDICATOR:L	1 - Success
:RUDAT:2B:STARTUPATT:INDICATOR:N	1 - Success

DLL Implementation:

```
Send_SCPI(":RUDAT:1A:STARTUPATT:INDICATOR:L", RetStr)
```

HTTP Implementation:

```
http://10.10.10.10/:RUDAT:1A:STARTUPATT:INDICATOR:L
```

#### See Also

[Save Counters & States](#)  
[Get Start-Up Attenuation Mode](#)  
[Set Start-Up Attenuation Value](#)  
[Get Start-Up Attenuation Value](#)

### 3.2 (d) - Get Start-Up Attenuation Mode

#### Description

Returns the start-up mode of a specific attenuator when the ZTM Series system is powered up. The attenuator can be configured to load the last remembered value, a specific fixed value or the maximum possible attenuation.

#### Command Syntax

**:STARTUPATT:INDICATOR?**

#### Return String

[mode]

Variable	Value	Description
[mode]	L	Last Value - the attenuator will load the last remembered attenuation See <a href="#">Save Counters &amp; States</a> for correct operation
	F	Fixed Value - the attenuator will load a fixed value (see Set Start-Up Attenuation Value)
	N	Normal - the attenuator will load the maximum possible attenuation (this is the default setting)

#### Examples

String to Send	String Returned
:RUDAT:1A:STARTUPATT:INDICATOR?	L
:RUDAT:2B:STARTUPATT:INDICATOR?	N

DLL Implementation:

```
Send_SCPI(":RUDAT:1A:STARTUPATT:INDICATOR?", RetStr)
```

HTTP Implementation:

```
http://10.10.10.10/:RUDAT:1A:STARTUPATT:INDICATOR?
```

#### See Also

[Save Counters & States](#)  
[Set Start-Up Attenuation Mode](#)  
[Set Start-Up Attenuation Value](#)  
[Get Start-Up Attenuation Value](#)

## 3.2 (e) - Set Start-Up Attenuation Value

### Description

Used in conjunction with [Set Start-Up Attenuation Mode](#) to set the start-up attenuation value for a specific attenuator. This is the initial attenuation setting that will be used when the ZTM Series system is powered up.

### Command Syntax

**:STARTUPATT:VALUE:** [att]

Variable	Description
[att]	The start-up attenuation setting to be loaded on power up (only valid when the attenuator "Start-Up Attenuation Mode" is set to mode "F" (fixed value).

### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed
	1 - Success	Command completed successfully

### Examples

String to Send	String Returned
:RUDAT:1A:STARTUPATT:VALUE:15	1 - Success
:RUDAT:2B:STARTUPATT:VALUE:20.25	1 - Success

DLL Implementation:

```
Send_SCPI(":RUDAT:1A:STARTUPATT:VALUE:15", RetStr)
```

HTTP Implementation:

```
http://10.10.10.10/:RUDAT:1A:STARTUPATT:VALUE:15
```

### See Also

[Set Start-Up Attenuation Mode](#)

[Get Start-Up Attenuation Mode](#)

[Get Start-Up Attenuation Value](#)

### 3.2 (f) - Get Start-Up Attenuation Value

#### Description

Returns the start-up attenuation value to be used for a specific attenuator when the ZTM Series system is powered on. This setting only applies when the attenuator "Start-Up Attenuation Mode" is set to mode "F" (fixed value).

#### Command Syntax

**:STARTUPATT:VALUE?**

#### Return String

**[mode]**

Variable	Value	Description
[mode]	L	Last Value - the attenuator will load the last remembered attenuation
	F	Fixed Value - the attenuator will load a fixed value (see Set Start-Up Attenuation Value)
	N	Normal - the attenuator will load the maximum possible attenuation (this is the default setting)

#### Examples

String to Send	String Returned
:RUDAT:1A:STARTUPATT:VALUE?	15.00
:RUDAT:2B:STARTUPATT:VALUE?	20.25

DLL Implementation:

```
Send_SCPI(":RUDAT:1A:STARTUPATT:VALUE?", RetStr)
```

HTTP Implementation:

```
http://10.10.10.10/:RUDAT:1A:STARTUPATT:VALUE?
```

#### See Also

[Set Start-Up Attenuation Mode](#)  
[Get Start-Up Attenuation Mode](#)  
[Set Start-Up Attenuation Value](#)

### 3.2 (g) - Get Maximum Attenuation

#### Description

Read the maximum possible attenuation setting of a specific programmable attenuator.

#### Command Syntax

`:MAX?`

#### Return String

`[value]`

Variable	Description
<code>[value]</code>	The maximum attenuation setting (dB) as a string of ASCII characters

#### Examples

String to Send	String Returned
<code>:RUDAT:1A:MAX?</code>	95.00
<code>:RUDAT:2B:MAX?</code>	95.00

DLL Implementation: `Send_SCPI(":RUDAT:1A:MAX?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:RUDAT:1A:MAX?`

#### See Also

[Set Attenuation](#)

[Get Attenuation](#)

### 3.3 - SPDT Switch Control

```
:SPDT:[location]:[command]:[value]
:SPDT:[location]:[query]?
```

The following commands and queries allow control of a specific SPDT switch within the modular test system. The component designator “SPDT” should be used along with the location of the switch within the system (for example “1A” or “1B” where there are 2 switches in window 1, or just “1” when there is only a single switch in window 1). The location “ALL” should be used when setting or querying the states of all switches.

	Description	Command/Query
a	Set SPDT Switch State	SPDT:[location]:STATE:[value]
b	Get SPDT Switch State	SPDT:[location]:STATE?
c	Set All SPDT Switch States	SPDT:ALL:STATE:[values]
d	Get All SPDT Switch States	SPDT:ALL:STATE?



### 3.3 (a) - Set SPDT Switch State

#### Description

Set the state of a specific SPDT switch.

#### Command Syntax

**:STATE:** [value]

Variable	Value	Description
[value]	1	Com port connected to port 1
	2	Com port connected to port 2

#### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed (switch not set)
	1 - Success	Command completed successfully

#### Examples

String to Send	String Returned
:SPDT:1A:STATE:2	1 - Success
:SPDT:2:STATE:1	1 - Success

DLL Implementation: `Send_SCPI(":SPDT:1A:STATE:2", RetStr)`

HTTP Implementation: `http://10.10.10.10/:SPDT:1A:STATE:2`

#### See Also

[Get SPDT Switch State](#)

[Set All SPDT Switch States](#)

[Get All SPDT Switch States](#)

### 3.3 (b) - Get SPDT Switch State

#### Description

Read the state of a specific SPDT switch.

#### Command Syntax

`:STATE?`

#### Return String

`[value]`

Variable	Value	Description
<code>[value]</code>	1	Com port connected to port 1
	2	Com port connected to port 2

#### Examples

String to Send	String Returned
<code>:SPDT:1A:STATE?</code>	2
<code>:SPDT:2:STATE?</code>	1

DLL Implementation: `Send_SCPI(":SPDT:1A:STATE?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:SPDT:1A:STATE?`

#### See Also

[Set SPDT Switch State](#)

[Set All SPDT Switch States](#)

[Get All SPDT Switch States](#)

### 3.3 (c) - Set All SPDT Switch States

#### Description

Simultaneously sets the state of all SPDT switches. The switch states are represented by a string of up to 12 characters for the ZTM Series and up to 6 characters for the RCM Series, with each character corresponding to an SPDT location in the test system, from location 1A, 1B, 2A, to 6B.

Note: The SPDT and transfer switches accept the same command arguments so a “Set All SPDT Switches” command would also set a transfer switch if a state is specified for a location containing a transfer switch.

#### Command Syntax

ZTM Series:

**:STATE:** [1A] [1B] [2A] [2B] [3A] [3B] [4A] [4B] [5A] [5B] [6A] [6B]

RCM Series:

**:STATE:** [1A] [1B] [2A] [2B] [3A] [3B]

Variable	Value	Description
[1A] to [6B]	1	Set SPDT state in this location to “Com to port 1”
	2	Set SPDT state in this location to “Com to port 2”
	x	Leave SPDT in this location unchanged / No SPDT in this location

Note: The string of switch states must always be supplied starting with location 1A but digits on the right-hand side can be omitted if the respective switch states are to be left unchanged or if the location does not contain an SPDT switch.

#### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed (switch not set)
	1 - Success	Command completed successfully

## Examples

The example ZTM Series configuration of figure 3 has SPDT switches in locations 2A, 2B, 4A and 4B so their states are represented by the 3<sup>rd</sup>, 4<sup>th</sup>, 7<sup>th</sup> and 8<sup>th</sup> characters in the string of switch states. All other characters should be “x” since those locations do not contain an SPDT switch.

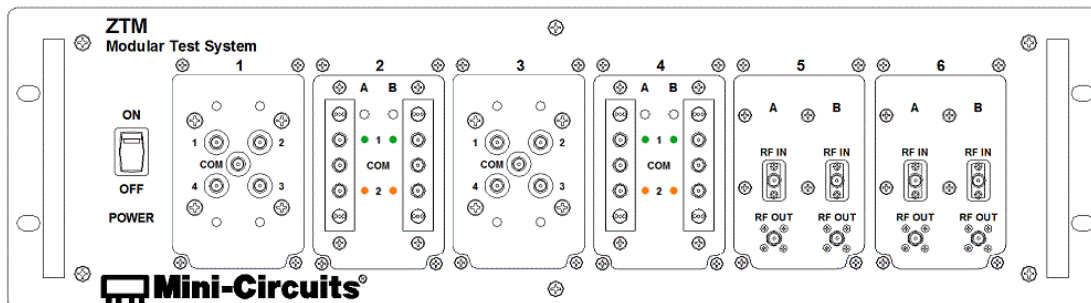


Fig 3 - Example configuration with SPDT switches in locations 2A, 2B, 4A and 4B

To set the SPDT switches in locations 2A and 2B to state 2, whilst leaving the states of the SPDT switches in locations 4A and 4B unchanged, the string to send is “xx22xxxxxxx”.

To set all 4 SPDT switches in locations 2A, 2B, 4A and 4B to states 2, 1, 2 and 1 respectively, the string to send is “xx21xx21xxx”.

Note: In both cases above, the trailing “x” characters (on the right-hand side of the string) can be omitted.

String to Send	String Returned
:SPDT:ALL:STATE:xx22	1 - Success
:SPDT:ALL:STATE:xx21xx21	1 - Success

DLL Implementation: `Send_SCPI(":SPDT:ALL:STATE:xx21xx21", RetStr)`  
 HTTP Implementation: `http://10.10.10.10/:SPDT:ALL:STATE:xx21xx21`

## See Also

[Set SPDT Switch State](#)  
[Get SPDT Switch State](#)  
[Get All SPDT Switch States](#)

### 3.3 (d) - Get All SPDT Switch States

#### Description

Returns the state of all SPDT switches. The switch states are represented by a string of 12 characters for the ZTM Series and 6 characters for the RCM Series, with each character corresponding to an SPDT location in the system, from location 1A, 1B, 2A, to 6B.

Note: The states of any transfer switches in the system will also be reported in the string.

#### Command Syntax

**:STATE?**

#### Return String

ZTM Series:

[1A] [1B] [2A] [2B] [3A] [3B] [4A] [4B] [5A] [5B] [6A] [6B]

RCM Series:

[1A] [1B] [2A] [2B] [3A] [3B]

Variable	Value	Description
[1A] to [6B]	1	SPDT state in this location is "Com to port 1"
	2	SPDT state in this location is "Com to port 2"
	x	No SPDT or transfer switch in this location

## Example

The example ZTM Series configuration of figure 4 has SPDT switches in locations 2A, 2B, 4A and 4B so their states are represented by the 3<sup>rd</sup>, 4<sup>th</sup>, 7<sup>th</sup> and 8<sup>th</sup> characters in the returned string of switch states. All other characters can be disregarded.

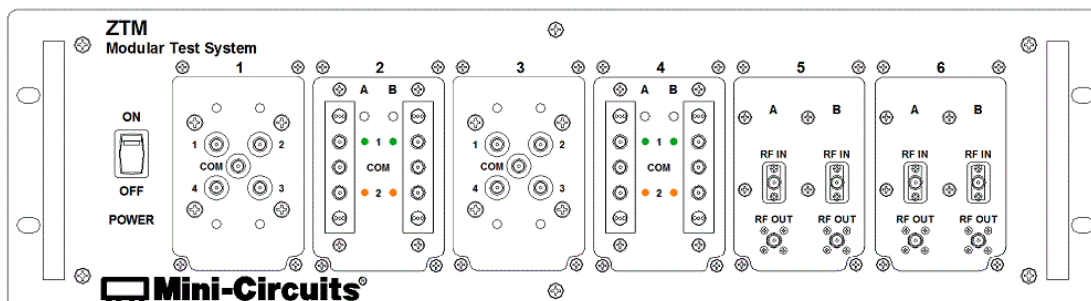


Fig 4 - Example configuration with SPDT switches in locations 2A, 2B, 4A and 4B

String to Send	String Returned
:SPDT:ALL:STATE?	xx21xx21xxxx

A return string of “xx21xx21xxxx” indicates the following SPDT switch states:

SPDT Window	State	Description
2A	2	Com connected to port 2
2B	1	Com connected to port 1
4A	2	Com connected to port 2
4B	1	Com connected to port 1

DLL Implementation: `Send_SCPI(":SPDT:ALL:STATE?", RetStr)`  
 HTTP Implementation: `http://10.10.10.10/:SPDT:ALL:STATE?`

## See Also

[Set SPDT Switch State](#)  
[Get SPDT Switch State](#)  
[Set All SPDT Switch States](#)

### 3.4 - SP4T Switch Control

```
:SP4T:[window]:[command]:[value]
:SP4T:[window]:[query]?
```

The following commands and queries allow control of a specific SP4T switch within the modular test system. The component designator “SP4T” should be used along with the location of the switch within system (windows 1 to 6). The location “ALL” should be used when setting or querying the states of all switches.

	Description	Command/Query
a	Set SP4T Switch State	SP4T:[window]:STATE:[value]
b	Get SP4T Switch State	SP4T:[window]:STATE?
c	Set All SP4T Switch States	SP4T:ALL:STATE:[values]
c	Get All SP4T Switch States	SP4T:ALL:STATE?



### 3.4 (a) - Set SP4T Switch State

#### Description

Set the state of a specific SP4T switch.

#### Command Syntax

**:STATE:** [value]

Variable	Value	Description
[value]	0	All ports disconnected
	1	Com port connected to port 1
	2	Com port connected to port 2
	3	Com port connected to port 3
	4	Com port connected to port 4

#### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed (switch not set)
	1 - Success	Command completed successfully

#### Examples

String to Send	String Returned
:SP4T:1:STATE:3	1 - Success
:SP4T:2:STATE:0	1 - Success

DLL Implementation: `Send_SCPI(":SP4T:1:STATE:3", RetStr)`

HTTP Implementation: `http://10.10.10.10/:SP4T:1:STATE:3`

#### See Also

[Get SP4T Switch State](#)  
[Set All SP4T Switch States](#)  
[Get All SP4T Switch States](#)

### 3.4 (b) - Get SP4T Switch State

#### Description

Read the state of a specific SP4T switch.

#### Command Syntax

**:STATE?**

#### Return String

**[value]**

Variable	Value	Description
[value]	0	All ports disconnected
	1	Com port connected to port 1
	2	Com port connected to port 2
	3	Com port connected to port 3
	4	Com port connected to port 4

#### Examples

String to Send	String Returned
:SP4T:1:STATE?	1
:SP4T:2:STATE?	0

DLL Implementation: `Send_SCPI(":SP4T:1:STATE?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:SP4T:1:STATE?`

#### See Also

[Set SP4T Switch State](#)

[Set All SP4T Switch States](#)

[Get All SP4T Switch States](#)

### 3.4 (c) - Set All SP4T Switch States

#### Description

Simultaneously sets the state of all SP4T switches. The switch states are represented by a string of up to 6 characters for the ZTM Series and up to 3 characters for the RCM Series, with each character corresponding to an SP4T location in the system.

#### Command Syntax

ZTM Series:

**:STATE:** [1] [2] [3] [4] [5] [6]

RCM Series:

**:STATE:** [1] [2] [3]

Variable	Value	Description
[1] to [6]	0	Disconnect all ports for the SP4T in this location
	1	Set SP4T state in this location to "Com to port 1"
	2	Set SP4T state in this location to "Com to port 2"
	3	Set SP4T state in this location to "Com to port 3"
	4	Set SP4T state in this location to "Com to port 4"
	x	Leave SP4T in this location unchanged / No SP4T in this location

Note: The string of switch states must always be supplied starting with window 1 but digits on the right-hand side can be omitted if the respective switch states are to be left unchanged or if the window does not contain an SP4T switch.

#### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed (switch not set)
	1 - Success	Command completed successfully

## Examples

The example ZTM Series configuration of figure 5 has SP4T switches in locations 1 and 3 so their states are represented by the 1<sup>st</sup> and 3<sup>rd</sup> characters in the string of switch states. All other characters should be “x” since those locations do not contain an SP4T switch.

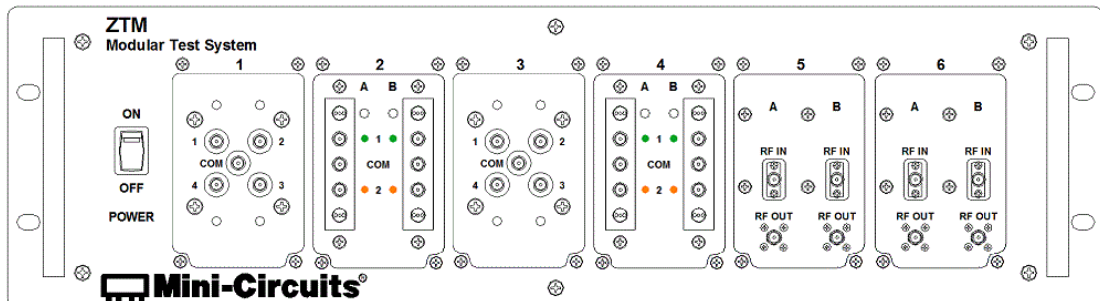


Fig 5 - Example configuration with SP4T switches in locations 1 and 3

To set the SP4T switch in location 1 to state 3, whilst leaving the state of the SP4T switch in location 3 unchanged, the string to send is “3xxxxx”.

To set both SP4T switches in to state 4, the string to send is “4x4xxx”.

Note: In both cases above, the trailing “x” characters (on the right-hand side of the string) can be omitted.

String to Send	String Returned
:SP4T:ALL:STATE:3	1 - Success
:SP4T:ALL:STATE:4x4	1 - Success

DLL Implementation: `Send_SCPI(":SP4T:ALL:STATE:4x4", RetStr)`  
 HTTP Implementation: `http://10.10.10.10/:SP4T:ALL:STATE:4x4`

## See Also

[Set SP4T Switch State](#)  
[Get SP4T Switch State](#)  
[Get All SP4T Switch States](#)

### 3.4 (d) - Get All SP4T Switch States

#### Description

Returns the state of all SP4T switches. The switch states are represented by a string of 6 characters for the ZTM Series and 3 characters for the RCM Series, with each character corresponding to an SP4T location in the system.

#### Command Syntax

**:STATE?**

#### Return String

ZTM Series:

[1] [2] [3] [4] [5] [6]

RCM Series:

[1] [2] [3]

Variable	Value	Description
[1] to [6]	0	All ports for the SP4T in this location disconnected
	1	SP4T state in this location is "Com to port 1"
	2	SP4T state in this location is "Com to port 2"
	3	SP4T state in this location is "Com to port 3"
	4	SP4T state in this location is "Com to port 4"
	x	No SP4T in this location

## Example

The example ZTM Series configuration of figure 6 has SP4T switches in windows 1 and 3 so their states are represented by the 1<sup>st</sup> and 3<sup>rd</sup> characters in the returned string of switch states. All other characters can be disregarded.

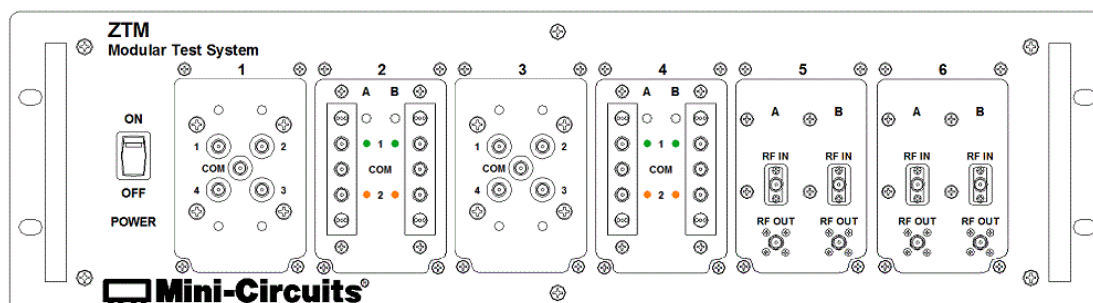


Fig 6 - Example configuration with SP4T switches in windows 1 and 3

String to Send	String Returned
:SP4T:ALL:STATE?	3x0xxx

A return string of “3x0xxx” indicates the following SP4T switch states:

SP4T Window	State	Description
1	3	Com connected to port 3
3	0	All ports disconnected

DLL Implementation: `Send_SCPI(":SP4T:ALL:STATE?", RetStr)`  
 HTTP Implementation: `http://10.10.10.10/:SP4T:ALL:STATE?`

## See Also

[Set SP4T Switch State](#)  
[Get SP4T Switch State](#)  
[Set All SP4T Switch States](#)

### 3.5 - SP6T Switch Control

```
:SP6T:[window]:[command]:[value]
:SP6T:[window]:[query]?
```

The following commands and queries allow control of a specific SP6T switch within the modular test system. The component designator “SP6T” should be used along with the location of the switch within the system. The location “ALL” should be used when setting or querying the states of all switches.

	Description	Command/Query
a	Set SP6T Switch State	SP6T:[window]:STATE:[value]
b	Get SP6T Switch State	SP6T:[window]:STATE?
c	Set All SP6T Switch States	SP6T:ALL:STATE:[values]
c	Get All SP6T Switch States	SP6T:ALL:STATE?

### 3.5 (a) - Set SP6T Switch State

#### Description

Set the state of a specific SP6T switch.

#### Command Syntax

**:STATE:** [value]

Variable	Value	Description
[value]	0	All ports disconnected
	1	Com port connected to port 1
	2	Com port connected to port 2
	3	Com port connected to port 3
	4	Com port connected to port 4
	5	Com port connected to port 5
	6	Com port connected to port 6

#### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed (switch not set)
	1 - Success	Command completed successfully

#### Examples

String to Send	String Returned
:SP6T:1:STATE:3	1 - Success
:SP6T:2:STATE:0	1 - Success

DLL Implementation: `Send_SCPI(":SP6T:1:STATE:3", RetStr)`

HTTP Implementation: `http://10.10.10.10/:SP6T:1:STATE:3`

#### See Also

[Get SP6T Switch State](#)  
[Set All SP6T Switch States](#)  
[Get All SP6T Switch States](#)



### 3.5 (b) - Get SP6T Switch State

#### Description

Read the state of a specific SP6T switch.

#### Command Syntax

**:STATE?**

#### Return String

**[value]**

Variable	Value	Description
[value]	0	All ports disconnected
	1	Com port connected to port 1
	2	Com port connected to port 2
	3	Com port connected to port 3
	4	Com port connected to port 4
	5	Com port connected to port 5
	6	Com port connected to port 6

#### Examples

String to Send	String Returned
:SP6T:1:STATE?	3
:SP6T:2:STATE?	0

DLL Implementation: `Send_SCPI(":SP6T:1:STATE?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:SP6T:1:STATE?`

#### See Also

[Set SP6T Switch State](#)

[Set All SP6T Switch States](#)

[Get All SP6T Switch States](#)

### 3.5 (c) - Set All SP6T Switch States

#### Description

Simultaneously sets the state of all SP6T switches. The switch states are represented by a string of up to 6 characters for the ZTM Series and up to 5 characters for the RCM Series, with each character corresponding to an SP6T location in the system.

#### Command Syntax

ZTM Series:

**:STATE:** [1] [2] [3] [4] [5] [6]

RCM Series:

**:STATE:** [1] [2] [3]

Variable	Value	Description
[1] to [6]	0	Disconnect all ports for the SP6T in this window
	1	Set SP6T state in this window to "Com to port 1"
	2	Set SP6T state in this window to "Com to port 2"
	3	Set SP6T state in this window to "Com to port 3"
	4	Set SP6T state in this window to "Com to port 4"
	5	Set SP6T state in this window to "Com to port 5"
	6	Set SP6T state in this window to "Com to port 6"
	x	Leave SP6T in this window unchanged / No SP6T in this window

Note: The string of switch states must always be supplied starting with window 1 but digits on the right-hand side can be omitted if the respective switch states are to be left unchanged or if the window does not contain an SP6T switch.

#### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed (switch not set)
	1 - Success	Command completed successfully

## Examples

The example ZTM Series configuration of figure 7 has SP6T switches in windows 1 and 3 so their states are represented by the 1<sup>st</sup> and 3<sup>rd</sup> characters in the string of switch states. All other characters should be “x” since those windows do not contain an SP6T switch.

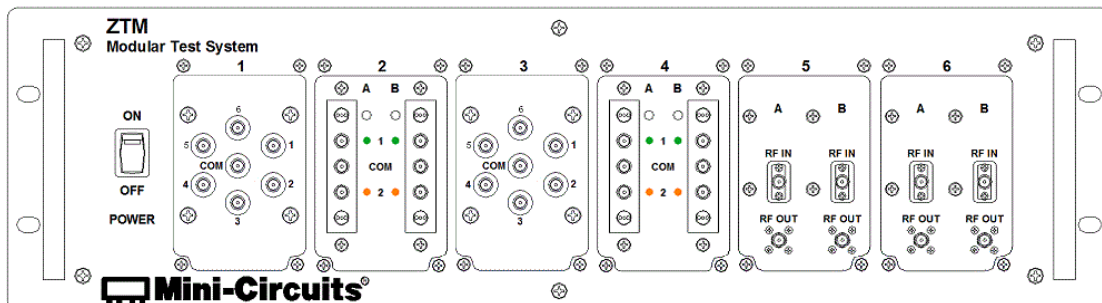


Fig 7 - Example configuration with SP6T switches in windows 1 and 3

To set the SP6T switch in window 1 to state 3, whilst leaving the state of the SP6T switch in window 3 unchanged, the string to send is “3xxxxx”.

To set both SP6T switches in to state 4, the string to send is “4x4xxx”.

Note: In both cases above, the trailing “x” characters (on the right-hand side of the string) can be omitted.

String to Send	String Returned
:SP6T:ALL:STATE:3	1 - Success
:SP6T:ALL:STATE:4x4	1 - Success

DLL Implementation: `Send_SCPI(":SP6T:ALL:STATE:4x4", RetStr)`

HTTP Implementation: `http://10.10.10.10/:SP6T:ALL:STATE:4x4`

## See Also

[Set SP6T Switch State](#)

[Get SP6T Switch State](#)

[Get All SP6T Switch States](#)

### 3.5 (d) - Get All SP6T Switch States

#### Description

Returns the state of all SP6T switches. The switch states are represented by a string of 6 characters for the ZTM Series and 3 characters for the RCM Series, with each character corresponding to an SP6T location.

#### Command Syntax

**:STATE?**

#### Return String

ZTM Series:

**[1] [2] [3] [4] [5] [6]**

RCM Series:

**[1] [2] [3]**

Variable	Value	Description
<b>[1] to [6]</b>	0	All ports for the SP6T in this window disconnected
	1	SP6T state in this window is "Com to port 1"
	2	SP6T state in this window is "Com to port 2"
	3	SP6T state in this window is "Com to port 3"
	4	SP6T state in this window is "Com to port 4"
	5	SP6T state in this window is "Com to port 5"
	6	SP6T state in this window is "Com to port 6"
	x	No SP6T in this window

## Example

The example ZTM Series configuration of figure 8 has SP6T switches in windows 1 and 3 so their states are represented by the 1<sup>st</sup> and 3<sup>rd</sup> characters in the returned string of switch states. All other characters can be disregarded.

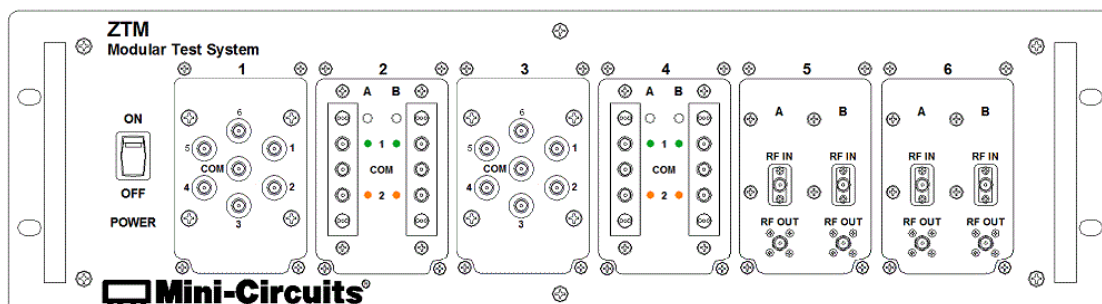


Fig 8 - Example configuration with SP6T switches in windows 1 and 3

String to Send	String Returned
:SP6T:ALL:STATE?	3x0xxx

A return string of "3x0xxx" indicates the following SP6T switch states:

SP6T Window	State	Description
1	3	Com connected to port 3
3	0	All ports disconnected

DLL Implementation: `Send_SCPI(":SP6T:ALL:STATE?", RetStr)`  
 HTTP Implementation: `http://10.10.10.10/:SP6T:ALL:STATE?`

## See Also

[Set SP6T Switch State](#)  
[Get SP6T Switch State](#)  
[Set All SP6T Switch States](#)

### 3.6 - SP8T Switch Control

```
:SP8T:[window]:[command]:[value]
:SP8T:[window]:[query]?
```

The following commands and queries allow control of a specific SP8T switch within the modular test system. The component designator “SP8T” should be used along with the location of the switch within the system. The location “ALL” should be used when setting or querying the states of all switches.

Note: SP8T switches are larger than the standard window size so the maximum number of fitted SP8T switches is 4 for the ZTM Series and 2 for the RCM Series.

	Description	Command/Query
a	Set SP8T Switch State	SP8T:[window]:STATE:[value]
b	Get SP8T Switch State	SP8T:[window]:STATE?
c	Set All SP8T Switch States	SP8T:ALL:STATE:[values]
c	Get All SP8T Switch States	SP8T:ALL:STATE?

### 3.6 (a) - Set SP8T Switch State

#### Description

Set the state of a specific SP8T switch.

#### Command Syntax

**:STATE:** [value]

Variable	Value	Description
[value]	0	All ports disconnected
	1	Com port connected to port 1
	2	Com port connected to port 2
	3	Com port connected to port 3
	4	Com port connected to port 4
	5	Com port connected to port 5
	6	Com port connected to port 6
	7	Com port connected to port 7
	8	Com port connected to port 8

#### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed (switch not set)
	1 - Success	Command completed successfully

#### Examples

String to Send	String Returned
:SP8T:1:STATE:3	1 - Success
:SP8T:2:STATE:0	1 - Success

DLL Implementation: `Send_SCPI(":SP8T:1:STATE:3", RetStr)`  
 HTTP Implementation: `http://10.10.10.10/:SP8T:1:STATE:3`

#### See Also

[Get SP8T Switch State](#)  
[Set All SP8T Switch States](#)  
[Get All SP8T Switch States](#)

### 3.6 (b) - Get SP8T Switch State

#### Description

Read the state of a specific SP8T switch.

#### Command Syntax

**:STATE?**

#### Return String

**[value]**

Variable	Value	Description
[value]	0	All ports disconnected
	1	Com port connected to port 1
	2	Com port connected to port 2
	3	Com port connected to port 3
	4	Com port connected to port 4
	5	Com port connected to port 5
	6	Com port connected to port 6
	7	Com port connected to port 7
	8	Com port connected to port 8

#### Examples

String to Send	String Returned
:SP8T:1:STATE?	3
:SP8T:2:STATE?	0

DLL Implementation: `Send_SCPI(":SP8T:1:STATE?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:SP8T:1:STATE?`

#### See Also

[Set SP8T Switch State](#)

[Set All SP8T Switch States](#)

[Get All SP8T Switch States](#)



### 3.6 (c) - Set All SP8T Switch States

#### Description

Simultaneously sets the state of all SP8T switches. The switch states are represented by a string of up to 4 characters for the ZTM Series and up to 2 characters for the RCM Series, with each character corresponding to an SP8T location in the system.

#### Command Syntax

ZTM Series:

**:STATE:** [1] [2] [3] [4]

RCM Series:

**:STATE:** [1] [2]

Variable	Value	Description
[1] to [4]	0	Disconnect all ports for the SP8T in this window
	1	Set SP8T state in this window to "Com to port 1"
	2	Set SP8T state in this window to "Com to port 2"
	3	Set SP8T state in this window to "Com to port 3"
	4	Set SP8T state in this window to "Com to port 4"
	5	Set SP8T state in this window to "Com to port 5"
	6	Set SP8T state in this window to "Com to port 6"
	7	Set SP8T state in this window to "Com to port 7"
	8	Set SP8T state in this window to "Com to port 8"
	x	Leave SP8T in this window unchanged / No SP8T in this window

Note: The string of switch states must always be supplied starting with window 1 but digits on the right-hand side can be omitted if the respective switch states are to be left unchanged or if the window does not contain an SP8T switch.

#### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed (switch not set)
	1 - Success	Command completed successfully

## Examples

ZTM-4SP8T-12 is fitted with 4 x SP8T switches in positions 1 to 4.

To set the SP8T switch in positions 1 and 3 to state 3, whilst leaving the state of the SP8T switches in position 2 and 3 unchanged, the string to send is "3x3".

To set all SP8T switches to state 4, the string to send is "4444".

Note: Trailing "x" characters (on the right-hand side of the string) can be omitted.

String to Send	String Returned
:SP8T:ALL:STATE:3x3	1 - Success
:SP8T:ALL:STATE:4444	1 - Success

DLL Implementation: `Send_SCPI(":SP8T:ALL:STATE:4444", RetStr)`

HTTP Implementation: `http://10.10.10.10/:SP8T:ALL:STATE:4444`

## See Also

[Set SP8T Switch State](#)

[Get SP8T Switch State](#)

[Get All SP8T Switch States](#)

### 3.6 (d) - Get All SP8T Switch States

#### Description

Returns the state of all SP8T switches. The switch states are represented by a string of 4 characters for the ZTM Series and 2 characters for the RCM Series, with each character corresponding to an SP8T location.

#### Command Syntax

**:STATE?**

#### Return String

ZTM Series:

**[1] [2] [3] [4]**

RCM Series:

**[1] [2]**

Variable	Value	Description
<b>[1] to [4]</b>	0	All ports for the SP8T in this window disconnected
	1	SP8T state in this window is "Com to port 1"
	2	SP8T state in this window is "Com to port 2"
	3	SP8T state in this window is "Com to port 3"
	4	SP8T state in this window is "Com to port 4"
	5	SP8T state in this window is "Com to port 5"
	6	SP8T state in this window is "Com to port 6"
	7	SP8T state in this window is "Com to port 7"
	8	SP8T state in this window is "Com to port 8"
	x	No SP8T in this window

## Example

ZTM-4SP8T-12 is fitted with 4 x SP8T switches in positions 1 to 4.

String to Send	String Returned
<a href="#">:SP8T:ALL:STATE?</a>	4422

A return string of “4422” indicates the following SP8T switch states:

SP8T Window	State	Description
1	4	Com connected to port 4
2	4	Com connected to port 4
3	2	Com connected to port 2
4	2	Com connected to port 2

DLL Implementation: [Send\\_SCPI\(":SP8T:ALL:STATE?", RetStr\)](#)

HTTP Implementation: <http://10.10.10.10/:SP8T:ALL:STATE?>

## See Also

[Set SP8T Switch State](#)

[Get SP8T Switch State](#)

[Set All SP8T Switch States](#)

### 3.7 - Transfer Switch Control

```
:MTS:[location]:[command]:[value]
:MTS:[location]:[query]?
```

The following commands and queries allow control of a specific transfer switch within the modular test system. The component designator “MTS” should be used along with the location of the switch within the system (for example “1A” or “1B” where there are 2 switches in location 1, or just “1” when there is only a single switch in location 1). The location “ALL” should be used when setting or querying the states of all switches.

	Description	Command/Query
a	Set Transfer Switch State	MTS:[location]:STATE:[value]
b	Get Transfer Switch State	MTS:[location]:STATE?
c	Set All Transfer Switch States	MTS:ALL:STATE:[values]
d	Get All Transfer Switch States	MTS:ALL:STATE?

### 3.7 (a) - Set Transfer Switch State

#### Description

Set the state of a specific transfer switch.

#### Command Syntax

**:STATE:** [value]

Variable	Value	Description
[value]	1	Transfer switch in state 1: <ul style="list-style-type: none"> <li>Port J1 connected to port J3</li> <li>Port J2 connected to port J4</li> </ul>
	2	Transfer switch in state 2: <ul style="list-style-type: none"> <li>Port J1 connected to port J2</li> <li>Port J3 connected to port J4</li> </ul>

#### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed (switch not set)
	1 - Success	Command completed successfully

#### Examples

String to Send	String Returned
:MTS:1A:STATE:2	1 - Success
:MTS:2:STATE:1	1 - Success

DLL Implementation: `Send_SCPI(":MTS:1A:STATE:2", RetStr)`  
HTTP Implementation: `http://10.10.10.10/:MTS:1A:STATE:2`

#### See Also

[Get Transfer Switch State](#)  
[Set All Transfer Switch States](#)  
[Get All Transfer Switch States](#)

### 3.7 (b) - Get Transfer Switch State

#### Description

Read the state of a specific transfer switch.

#### Command Syntax

**:STATE?**

#### Return String

**[value]**

Variable	Value	Description
[value]	1	Transfer switch in state 1: <ul style="list-style-type: none"> <li>Port J1 connected to port J3</li> <li>Port J2 connected to port J4</li> </ul>
	2	Transfer switch in state 2: <ul style="list-style-type: none"> <li>Port J1 connected to port J2</li> <li>Port J3 connected to port J4</li> </ul>

#### Examples

String to Send	String Returned
:MTS:1A:STATE?	1
:MTS:2:STATE?	0

DLL Implementation: `Send_SCPI(":MTS:1A:STATE?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:MTS:1A:STATE?`

#### See Also

[Set Transfer Switch State](#)

[Set All Transfer Switch States](#)

[Get All Transfer Switch States](#)

### 3.7 (c) - Set All Transfer Switch States

#### Description

Simultaneously sets the state of all transfer switches. The switch states are represented by a string of up to 12 characters for the ZTM Series or up to 6 characters for the RCM Series, with each character corresponding to a transfer switch location within the system.

Note: The SPDT and transfer switches accept the same command arguments so a “Set All Transfer Switches” command would also set an SPDT switch if a state is specified for a location containing an SPDT switch.

#### Command Syntax

ZTM Series:

**:STATE:** [1A] [1B] [2A] [2B] [3A] [3B] [4A] [4B] [5A] [5B] [6A] [6B]

RCM Series:

**:STATE:** [1A] [1B] [2A] [2B] [3A] [3B]

Variable	Value	Description
[1A] to [6B]	1	Set transfer switch to state 1: <ul style="list-style-type: none"> <li>Port J1 connected to port J3</li> <li>Port J2 connected to port J4</li> </ul>
	2	Set transfer switch to state 2: <ul style="list-style-type: none"> <li>Port J1 connected to port J2</li> <li>Port J3 connected to port J4</li> </ul>
	x	Leave transfer switch state in this location unchanged / No transfer switch in this location

Note: The string of switch states must always be supplied starting with location 1A but digits on the right-hand side can be omitted if the respective switch states are to be left unchanged or if the location does not contain a transfer switch.

#### Return String

[status]

Variable	Value	Description
[status]	0 - Failed	Command failed (switch not set)
	1 - Success	Command completed successfully



## Examples

The example ZTM Series configuration of figure 9 has transfer switches in locations 2A, 2B, 4A and 4B so their states are represented by the 3<sup>rd</sup>, 4<sup>th</sup>, 7<sup>th</sup> and 8<sup>th</sup> characters in the string of switch states. All other characters should be “x” since those locations do not contain a transfer switch.

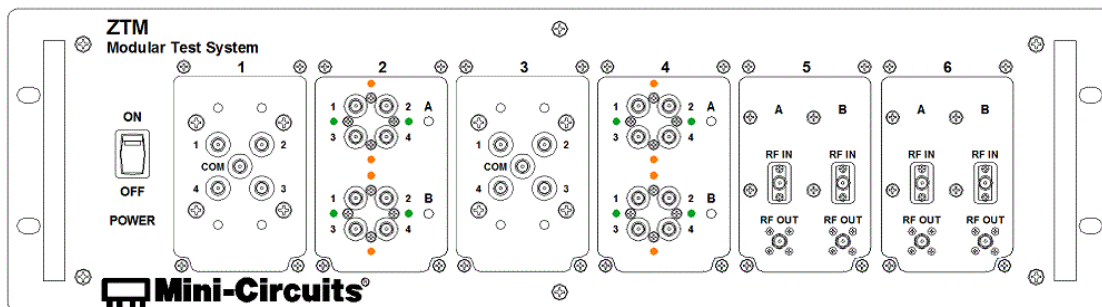


Fig 9 - Example configuration with transfer switches in locations 2A, 2B, 4A and 4B

To set the transfer switches in locations 2A and 2B to state 1, whilst leaving the states of the transfer switches in locations 4A and 4B unchanged, the string to send is “xx11xxxxxxx”.

To set all 4 transfer switches in locations 2A, 2B, 4A and 4B to states 2, 2, 2 and 1 respectively, the string to send is “xx22xx21xxxx”.

Note: In both cases above, the trailing “x” characters (on the right-hand side of the string) can be omitted.

String to Send	String Returned
:MTS:ALL:STATE:xx11	1 - Success
:MTS:ALL:STATE:xx22xx21	1 - Success

DLL Implementation: `Send_SCPI(":MTS:ALL:STATE:xx22xx21", RetStr)`  
 HTTP Implementation: `http://10.10.10.10/:MTS:ALL:STATE:xx22xx21`

## See Also

[Set Transfer Switch State](#)  
[Get Transfer Switch State](#)  
[Get All Transfer Switch States](#)

### 3.7 (d) - Get All Transfer Switch States

#### Description

Returns the state of all transfer switches. The switch states are represented by a string of 12 characters for the ZTM Series or 6 characters for the RCM Series, with each character corresponding to a transfer switch slot sub-location in the system.

Note: The states of any SPDT switches in the ZTM Series system will also be reported in the string.

#### Command Syntax

**: STATE?**

#### Return String

ZTM Series:

[1A] [1B] [2A] [2B] [3A] [3B] [4A] [4B] [5A] [5B] [6A] [6B]

RCM Series:

[1A] [1B] [2A] [2B] [3A] [3B]

Variable	Value	Description
[1A] to [6B]	1	Transfer switch state in this location is 1: <ul style="list-style-type: none"> <li>Port J1 connected to port J3</li> <li>Port J2 connected to port J4</li> </ul>
	2	Transfer switch state in this location is 2: <ul style="list-style-type: none"> <li>Port J1 connected to port J2</li> <li>Port J3 connected to port J4</li> </ul>
	x	No transfer or SPDT switch in this location

## Example

The example ZTM Series configuration of figure 10 has transfer switches in slots 2A, 2B, 4A and 4B so their states are represented by the 3<sup>rd</sup>, 4<sup>th</sup>, 7<sup>th</sup> and 8<sup>th</sup> characters in the returned string of switch states. All other characters can be disregarded.

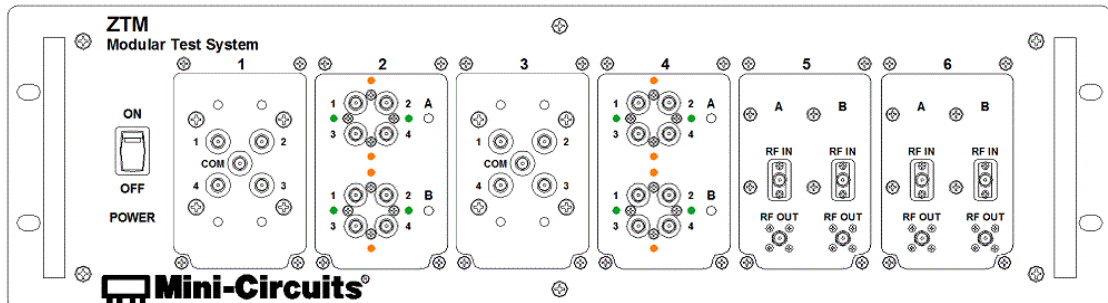


Fig 10 - Example configuration with transfer switches in slots 2A, 2B, 4A and 4B

String to Send	String Returned
:MTS:ALL:STATE?	xx12xx11xxxx

A return string of "xx12xx11xxxx" indicates the following transfer switch states:

MTS Window	State	Description
2A	1	<ul style="list-style-type: none"> <li>Port J1 connected to port J3</li> <li>Port J2 connected to port J4</li> </ul>
2B	2	<ul style="list-style-type: none"> <li>Port J1 connected to port J2</li> <li>Port J3 connected to port J4</li> </ul>
4A	1	<ul style="list-style-type: none"> <li>Port J1 connected to port J3</li> <li>Port J2 connected to port J4</li> </ul>
4B	1	<ul style="list-style-type: none"> <li>Port J1 connected to port J3</li> <li>Port J2 connected to port J4</li> </ul>

DLL Implementation: `Send_SCPI(":MTS:ALL:STATE?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:MTS:ALL:STATE?`

## See Also

[Set Transfer Switch State](#)

[Get Transfer Switch State](#)

[Set All Transfer Switch States](#)

### 3.8 - Switch Start-Up and Counter Properties

```
:SPDT:[location]:[command]:[value]
:SP4T:[location]:[command]:[value]
:SP6T:[location]:[command]:[value]
:MTS:[location]:[command]:[value]
```

The following commands and queries enable the user to work with the start-up settings and counter for any given switch in the modular test system. The component designator “SPDT”, “SP4T”, or “MTS” should be used along with the location of the switch within the system.

	Description	Command/Query
a	Set Switch Start-Up Mode	STARTUPSW:INDICATOR:[mode]
b	Get Switch Start-Up Mode	STARTUPSW:INDICATOR?
c	Get Switch Counter	SCOUNTER?

### 3.8 (a) - Set Switch Start-Up Mode

#### Description

Sets the start-up state for a specific switch when the system is powered up.

#### Command Syntax

**:STARTUPSW:INDICATOR:[mode]**

Variable	Value	Description
[mode]	L	Last Value - the switch will power up with the last remembered switch state See <a href="#">Save Counters &amp; States</a> for correct operation
	N	Normal - the switch will power up in the default state: <ul style="list-style-type: none"> <li>SPDT: Com port connected to port 1</li> <li>SP4T / SP6T / SP8T: All ports disconnected</li> <li>Transfer: J1 &lt;&gt; J3; J2 &lt;&gt; J4</li> </ul>

#### Return String

**[status]**

Variable	Value	Description
[status]	0 - Failed	Command failed
	1 - Success	Command completed successfully

#### Examples

String to Send	String Returned
:SPDT:1A:STARTUPSW:INDICATOR:L	1 - Success
:SP4T:1:STARTUPSW:INDICATOR:L	1 - Success
:SP6T:1:STARTUPSW:INDICATOR:L	1 - Success
:MTS:1A:STARTUPSW:INDICATOR:L	1 - Success

#### DLL Implementation:

```
Send_SCPI(":SPDT:1A:STARTUPSW:INDICATOR:L", RetStr)
Send_SCPI(":SP4T:1:STARTUPSW:INDICATOR:L", RetStr)
Send_SCPI(":SP6T:1:STARTUPSW:INDICATOR:L", RetStr)
Send_SCPI(":MTS:1A:STARTUPSW:INDICATOR:L", RetStr)
```

#### HTTP Implementation:

```
http://10.10.10.10/:SPDT:1A:STARTUPSW:INDICATOR:L
http://10.10.10.10/:SP4T:1:STARTUPSW:INDICATOR:L
http://10.10.10.10/:SP6T:1:STARTUPSW:INDICATOR:L
http://10.10.10.10/:MTS:1A:STARTUPSW:INDICATOR:L
```

#### See Also

[Save Counters & States](#)  
[Get Switch Start-Up Mode](#)  
[Get Switch Counter](#)

### 3.8 (b) - Get Switch Start-Up Mode

#### Description

Returns the start-up state that a specific switch will use when the system is powered up.

#### Command Syntax

**:STARTUPSW:INDICATOR?**

#### Return String

**[mode]**

Variable	Value	Description
[mode]	L	Last Value - the switch will power up with the last remembered switch state See <a href="#">Save Counters &amp; States</a> for correct operation
	N	Normal - the switch will power up in the default state: <ul style="list-style-type: none"> <li>SPDT: Com port connected to port 1</li> <li>SP4T / SP6T / SP8T: All ports disconnected</li> <li>Transfer: J1 &lt;&gt; J3; J2 &lt;&gt; J4</li> </ul>

#### Examples

String to Send	String Returned
:SPDT:1A:STARTUPATT:INDICATOR?	L
:SP4T:1:STARTUPATT:INDICATOR?	L
:SP6T:1:STARTUPATT:INDICATOR?	L
:MTS:1A:STARTUPATT:INDICATOR?	L

#### DLL Implementation:

```
Send_SCPI(":SPDT:1A:STARTUPSW:INDICATOR?", RetStr)
Send_SCPI(":SP4T:1:STARTUPSW:INDICATOR?", RetStr)
Send_SCPI(":SP6T:1:STARTUPSW:INDICATOR?", RetStr)
Send_SCPI(":MTS:1A:STARTUPSW:INDICATOR?", RetStr)
```

#### HTTP Implementation:

```
http://10.10.10.10/:SPDT:1A:STARTUPSW:INDICATOR?
http://10.10.10.10/:SP4T:1:STARTUPSW:INDICATOR?
http://10.10.10.10/:SP6T:1:STARTUPSW:INDICATOR?
http://10.10.10.10/:MTS:1A:STARTUPSW:INDICATOR?
```

#### See Also

[Save Counters & States](#)  
[Set Switch Start-Up Mode](#)  
[Get Switch Counter](#)

### 3.8 (c) - Get Switch Counter

#### Description

Returns a counter value indicating the number of switching cycles undertaken in the lifetime of a specific switch.

Note: See [Save Counters & States](#) for correct operation.

#### Command Syntax

**: SCOUNTER?**

#### Return String (SPDT and MTS)

**[count]**

Variable	Description
[count]	The number of switch cycles undertaken in the lifetime of the specified switch

#### Return String (SP4T and SP6T)

**[count1] ; [count2] ; [count3] ; [count4]**

Variable	Description
[count1]	The number of connections to port 1 undertaken in the lifetime of the specified switch
[count2]	The number of connections to port 2 undertaken in the lifetime of the specified switch
[count3]	The number of connections to port 3 undertaken in the lifetime of the specified switch
[count4]	The number of connections to port 4 undertaken in the lifetime of the specified switch
[count5]	SP6T only. The number of connections to port 5 undertaken in the lifetime of the specified switch
[count6]	SP6T only. The number of connections to port 6 undertaken in the lifetime of the specified switch

## Examples

String to Send	String Returned
<a href="#">:SPDT:1A:SCOUNTER?</a>	9540
<a href="#">:SP4T:1:SCOUNTER?</a>	2000;1253;1500;1685
<a href="#">:SP6T:1:SCOUNTER?</a>	195;452;300;125;850;647
<a href="#">:MTS:1A:SCOUNTER?</a>	9540

DLL Implementation:

```
Send_SCPI(":SPDT:1A:SCOUNTER?", RetStr)
Send_SCPI(":SP4T:1:SCOUNTER?", RetStr)
Send_SCPI(":SP6T:1:SCOUNTER?", RetStr)
Send_SCPI(":MTS:1A:SCOUNTER?", RetStr)
```

HTTP Implementation:

```
http://10.10.10.10/:SPDT:1A:SCOUNTER?
http://10.10.10.10/:SP4T:1:SCOUNTER?
http://10.10.10.10/:SP6T:1:SCOUNTER?
http://10.10.10.10/:MTS:1A:SCOUNTER?
```

## See Also

[Save Counters & States](#)  
[Set Switch Start-Up Mode](#)  
[Get Switch Start-Up Mode](#)



### 3.9 - Component Labels

These commands enable the user to set a custom label for easy identification on any given component in the modular test system. The component designator is not needed so the full SCPI commands are as below.

	Description	Command/Query
<b>a</b>	Set Component Label	<code>:LABEL:[location]: "[text]"</code>
<b>b</b>	Get Component Label	<code>:LABEL:[location]?</code>

#### 3.9 (a) - Set Component Label

##### Description

Set a custom label for easy identification of a specific component.

##### Command Syntax

`:LABEL:[location]: "[text]"`

Variable	Description
<code>[text]</code>	A string of ASCII characters to be set as an easy identification label for the component. Up to 24 characters allowed.

##### Return String

`[status]`

Variable	Value	Description
<code>[status]</code>	0 - Failed	Command failed (switch not set)
	1 - Success	Command completed successfully

##### Examples

String to Send	String Returned
<code>:LABEL:1A:"Input_SPDT_1"</code>	1 - Success
<code>:LABEL:1:"Input_SP4T_1"</code>	1 - Success

DLL Implementation: `Send_SCPI(":LABEL:1A:"Input_SPDT_1", RetStr)`

HTTP Implementation: `http://10.10.10.10/:LABEL:1A:"Input_SPDT_1"`

##### See Also

[Get Component Label](#)

### 3.9 (b) - Get Component Label

#### Description

Get the custom label of a specific component (used for easy identification).

#### Command Syntax

```
:LABEL: [location]?
```

#### Return String

```
LABEL=" [text] "
```

Variable	Description
[text]	The component label as a string of up to 24 ASCII characters

#### Examples

String to Send	String Returned
:LABEL:1A?	LABEL="Input_SPDT_1"
:LABEL:1?	LABEL="Input_SP4T_1"

DLL Implementation: `Send_SCPI(":LABEL:1A?", RetStr)`

HTTP Implementation: `http://10.10.10.10/:LABEL:1A?`

#### See Also

[Set Component Label](#)

### 3.10 - SCPI - Ethernet Configuration Commands

These functions apply to RCM and ZTM Series models with firmware C2 or later.

	Description	Command/Query
a	Set Static IP Address	:ETHERNET:CONFIG:IP:[ip]
b	Get Static IP Address	:ETHERNET:CONFIG:IP?
c	Set Static Subnet Mask	:ETHERNET:CONFIG:SM:[mask]
d	Get Static Subnet Mask	:ETHERNET:CONFIG:SM?
e	Set Static Network Gateway	:ETHERNET:CONFIG:NG:[gateway]
f	Get Static Network Gateway	:ETHERNET:CONFIG:NG?
g	Set HTTP Port	:ETHERNET:CONFIG:HTPORT:[port]
h	Get HTTP Port	:ETHERNET:CONFIG:HTPORT?
i	Set Telnet Port	:ETHERNET:CONFIG:TELNETPORT:[port]
j	Get Telnet Port	:ETHERNET:CONFIG:TELNETPORT?
k	Set Password Requirement	:ETHERNET:CONFIG:PWDENABLED:[enabled]
l	Get Password Requirement	:ETHERNET:CONFIG:PWDENABLED?
m	Set Password	:ETHERNET:CONFIG:PWD:[pwd]
n	Get Password	:ETHERNET:CONFIG:PWD?
o	Set DHCP Status	:ETHERNET:CONFIG:DHCPENABLED:[enabled]
p	Get DHCP Status	:ETHERNET:CONFIG:DHCPENABLED?
q	Get MAC Address	:ETHERNET:CONFIG:MAC?
r	Get Current Ethernet Configuration	:ETHERNET:CONFIG:LISTEN?
s	Update Ethernet Settings	:ETHERNET:CONFIG:INIT

### 3.10 (a) - Set Static IP Address

#### Description

Sets the IP address to be used by the system for Ethernet communication when using static IP settings. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use. Changes to the Ethernet configuration only take effect after the [Update Ethernet Settings](#) command has been issued.

#### Requirements

Firmware C2 or later.

#### Command Syntax

```
:ETHERNET:CONFIG:IP:[ip]
```

Variable	Description
[ip]	The static IP address to be used by the system; must be valid and available on the network

#### Return String

```
[status]
```

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

#### Examples

String to Send	String Returned
:ETHERNET:CONFIG:IP:192.100.1.1	1

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:IP:192.100.1.1
```

#### See Also

[Get Static IP Address](#)  
[Set Static Subnet Mask](#)  
[Set Static Network Gateway](#)  
[Update Ethernet Settings](#)

### 3.10 (b) - Get Static IP Address

#### Description

Returns the IP address to be used by the system for Ethernet communication when static IP settings are in use. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use.

#### Requirements

Firmware C2 or later.

#### Command Syntax

`:ETHERNET:CONFIG:IP?`

#### Return String

`[ip]`

Variable	Description
<code>[ip]</code>	The static IP address to be used by the system

#### Examples

String to Send	String Returned
<code>:ETHERNET:CONFIG:IP?</code>	<code>192.100.1.1</code>

HTTP Implementation:

`http://10.10.10.10/:ETHERNET:CONFIG:IP?`

#### See Also

[Set Static IP Address](#)  
[Get Static Subnet Mask](#)  
[Get Static Network Gateway](#)  
[Get Current Ethernet Configuration](#)

### 3.10 (c) - Set Static Subnet Mask

#### Description

Sets the subnet mask to be used by the system for Ethernet communication when using static IP settings. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use. Changes to the Ethernet configuration only take effect after the [Update Ethernet Settings](#) command has been issued.

#### Requirements

Firmware C2 or later.

#### Command Syntax

```
:ETHERNET:CONFIG:SM:[mask]
```

Variable	Description
[mask]	The subnet mask for communication on the network

#### Return String

```
[status]
```

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

#### Examples

String to Send	String Returned
:ETHERNET:CONFIG:SM:255.255.255.0	1

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:SM:255.255.255.0
```

#### See Also

[Set Static IP Address](#)  
[Get Static Subnet Mask](#)  
[Set Static Network Gateway](#)  
[Update Ethernet Settings](#)

### 3.10 (d) - Get Static Subnet Mask

#### Description

Returns the subnet mask to be used by the system for Ethernet communication when static IP settings are in use. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use.

#### Requirements

Firmware C2 or later.

#### Command Syntax

`:ETHERNET:CONFIG:SM?`

#### Return String

`[mask]`

Variable	Description
<code>[mask]</code>	The subnet mask for communication on the network

#### Examples

String to Send	String Returned
<code>:ETHERNET:CONFIG:SM?</code>	255.255.255.0

HTTP Implementation:

`http://10.10.10.10/:ETHERNET:CONFIG:SM?`

#### See Also

[Get Static IP Address](#)  
[Set Static Subnet Mask](#)  
[Get Static Network Gateway](#)  
[Get Current Ethernet Configuration](#)

### 3.10 (e) - Set Static Network Gateway

#### Description

Sets the IP address of the network gateway to be used by the system for Ethernet communication when using static IP settings. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use. Changes to the Ethernet configuration only take effect after the [Update Ethernet Settings](#) command has been issued.

#### Requirements

Firmware C2 or later.

#### Command Syntax

```
:ETHERNET:CONFIG:NG:[gateway]
```

Variable	Description
[gateway]	IP address of the network gateway

#### Return String

```
[status]
```

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

#### Examples

String to Send	String Returned
:ETHERNET:CONFIG:NG:192.100.1.0	1

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:NG:192.168.100.1.0
```

#### See Also

[Set Static IP Address](#)  
[Set Static Subnet Mask](#)  
[Get Static Network Gateway](#)  
[Update Ethernet Settings](#)



### 3.10 (f) - Get Static Network Gateway

#### Description

Returns the IP address of the network gateway to be used by the system for Ethernet communication when static IP settings are in use. DHCP must be disabled for this setting to apply, otherwise a dynamic IP address will be in use.

#### Requirements

Firmware C2 or later.

#### Command Syntax

`:ETHERNET:CONFIG:NG?`

#### Return String

`[gateway]`

Variable	Description
<code>[gateway]</code>	IP address of the network gateway

#### Examples

String to Send	String Returned
<code>:ETHERNET:CONFIG:NG?</code>	<code>192.168.1.0</code>

HTTP Implementation:

`http://10.10.10.10/:ETHERNET:CONFIG:NG?`

#### See Also

[Get Static IP Address](#)  
[Get Static Subnet Mask](#)  
[Set Static Network Gateway](#)  
[Get Current Ethernet Configuration](#)

### 3.10 (g) - Set HTTP Port

#### Description

Sets the IP port to be used for HTTP communication. Changes to the Ethernet configuration only take effect after the [Update Ethernet Settings](#) command has been issued.

#### Requirements

Firmware C2 or later.

#### Command Syntax

```
:ETHERNET:CONFIG:HTPORT:[port]
```

Variable	Description
[port]	IP port to be used for HTTP communication. The port will need to be included in all HTTP commands if any other than the default port 80 is selected.

#### Return String

```
[status]
```

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

#### Examples

String to Send	String Returned
:ETHERNET:CONFIG:HTPORT:8080	1

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:HTPORT:8080
```

#### See Also

[Get HTTP Port](#)  
[Set Telnet Port](#)  
[Update Ethernet Settings](#)

### 3.10 (h) - Get HTTP Port

#### Description

Gets the IP port to be used for HTTP communication.

#### Requirements

Firmware C2 or later.

#### Command Syntax

```
:ETHERNET:CONFIG:HTPORT?
```

#### Return String

```
[port]
```

Variable	Description
[port]	IP port to be used for HTTP communication

#### Examples

String to Send	String Returned
:ETHERNET:CONFIG:HTPORT?	8080

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:HTPORT?
```

#### See Also

[Set HTTP Port](#)  
[Get Telnet Port](#)

### 3.10 (i) - Set Telnet Port

#### Description

Sets the IP port to be used for Telnet communication. Changes to the Ethernet configuration only take effect after the [Update Ethernet Settings](#) command has been issued.

#### Requirements

Firmware C2 or later.

#### Command Syntax

```
:ETHERNET:CONFIG:TELNETPORT:[port]
```

Variable	Description
[port]	IP port to be used for Telnet communication. The port will need to be included when initiating a Telnet session if other than the default port 23 is selected.

#### Return String

```
[status]
```

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

#### Examples

String to Send	String Returned
:ETHERNET:CONFIG:TELNETPORT:21	1

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:TELNETPORT:21
```

#### See Also

[Set HTTP Port](#)  
[Get Telnet Port](#)  
[Update Ethernet Settings](#)

### 3.10 (j) - Get Telnet Port

#### Description

Gets the IP port to be used for Telnet communication.

#### Requirements

Firmware C2 or later.

#### Command Syntax

```
:ETHERNET:CONFIG:TELNETPORT?
```

#### Return String

```
[port]
```

Variable	Description
[port]	IP port to be used for Telnet communication

#### Examples

String to Send	String Returned
:ETHERNET:CONFIG:TELNETPORT?	1

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:TELNETPORT?
```

#### See Also

[Get HTTP Port](#)  
[Set Telnet Port](#)

### 3.10 (k) - Set Password Requirement

#### Description

Sets whether or not a password is required for Ethernet communication. Changes to the Ethernet configuration only take effect after the [Update Ethernet Settings](#) command has been issued.

#### Requirements

Firmware C2 or later.

#### Command Syntax

```
:ETHERNET:CONFIG:PWDENABLED:[enabled]
```

Variable	Value	Description
[enabled]	0	Password not required for Ethernet communication
	1	Password required for Ethernet communication

#### Return String

```
[status]
```

Variable	Value	Description
[status]	0	Command failed
	1	Command completed successfully

#### Examples

String to Send	String Returned
:ETHERNET:CONFIG:PWDENABLED:1	1

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:PWDENABLED:1
```

#### See Also

[Get Password Requirement](#)  
[Set Password](#)  
[Get Password](#)  
[Update Ethernet Settings](#)

### 3.10 (I) - Get Password Requirement

#### Description

Indicates whether or not a password is required for Ethernet communication.

#### Requirements

Firmware C2 or later.

#### Command Syntax

`:ETHERNET:CONFIG:PWDENABLED?`

#### Return String

`[enabled]`

Variable	Value	Description
<code>[enabled]</code>	0	Password not required for Ethernet communication
	1	Password required for Ethernet communication

#### Examples

String to Send	String Returned
<code>:ETHERNET:CONFIG:PWDENABLED?</code>	1

HTTP Implementation:

`http://10.10.10.10/:ETHERNET:CONFIG:PWDENABLED?`

#### See Also

[Set Password Requirement](#)  
[Set Password](#)  
[Get Password](#)

### 3.10 (m) - Set Password

#### Description

Sets the password for Ethernet communication. The password will only be required for communication with the device when password security is enabled. Changes to the Ethernet configuration only take effect after the [Update Ethernet Settings](#) command has been issued.

#### Requirements

Firmware C2 or later.

#### Command Syntax

`:ETHERNET:CONFIG:PWD:[pwd]`

Variable	Description
<code>[pwd]</code>	Password to set for Ethernet communication (not case sensitive)

#### Return String

`[status]`

Variable	Value	Description
<code>[status]</code>	0	Command failed
	1	Command completed successfully

#### Examples

String to Send	String Returned
<code>:ETHERNET:CONFIG:PWD:PASS-123</code>	1

HTTP Implementation:

<http://10.10.10.10/:ETHERNET:CONFIG:PWD:PASS-123>

#### See Also

[Set Password Requirement](#)  
[Get Password Requirement](#)  
[Get Password](#)  
[Update Ethernet Settings](#)



### 3.10 (n) - Get Password

#### Description

Returns the password for Ethernet communication. The password will only be required for communication with the device when password security is enabled

#### Requirements

Firmware C2 or later.

#### Command Syntax

`:ETHERNET:CONFIG:PWD?`

#### Return String

[pwd]

Variable	Description
[pwd]	Password for Ethernet communication (not case sensitive)

#### Examples

String to Send	String Returned
<code>:ETHERNET:CONFIG:PWD?</code>	<code>PASS-123</code>

HTTP Implementation:

`http://10.10.10.10/:ETHERNET:CONFIG:PWD?`

#### See Also

[Set Password Requirement](#)  
[Get Password Requirement](#)  
[Set Password](#)

### 3.10 (o) - Set DHCP Status

#### Description

Enables or disables DHCP (Dynamic Host Control Protocol). When enabled the system will request a valid IP address from the network's DHCP server. When disabled, the system's static IP settings will be used. Changes to the Ethernet configuration only take effect after the [Update Ethernet Settings](#) command has been issued.

#### Requirements

Firmware C2 or later.

#### Command Syntax

`:ETHERNET:CONFIG:DHCPENABLED:[enabled]`

Variable	Value	Description
<code>[enabled]</code>	0	DHCP disabled (static IP settings will be used)
	1	DHCP enabled (IP address will be requested from DHCP server on the network)

#### Return String

`[status]`

Variable	Value	Description
<code>[status]</code>	0	Command failed
	1	Command completed successfully

#### Examples

String to Send	String Returned
<code>:ETHERNET:CONFIG:DHCPENABLED:1</code>	1

HTTP Implementation:

<http://10.10.10.10/:ETHERNET:CONFIG:DHCPENABLED:1>

#### See Also

[Set Static IP Address](#)

[Get DHCP Status](#)

[Update Ethernet Settings](#)

### 3.10 (p) - Get DHCP Status

#### Description

Indicates whether or not DHCP (Dynamic Host Control Protocol) is enabled. When enabled the system will request a valid IP address from the network's DHCP server. When disabled, the system's static IP settings will be used.

#### Requirements

Firmware C2 or later.

#### Command Syntax

`:ETHERNET:CONFIG:DHCPENABLED?`

#### Return String

`[enabled]`

Variable	Value	Description
<code>[enabled]</code>	0	DHCP disabled (static IP settings will be used)
	1	DHCP enabled (IP address will be requested from DHCP server on the network)

#### Examples

String to Send	String Returned
<code>:ETHERNET:CONFIG:DHCPENABLED?</code>	1

HTTP Implementation:

`http://10.10.10.10/:ETHERNET:CONFIG:DHCPENABLED?`

#### See Also

[Set Static IP Address](#)

[Set DHCP Status](#)

[Get Current Ethernet Configuration](#)

### 3.10 (q) - Get MAC Address

#### Description

Returns the MAC (Media Access Control) address of the system (a physical hardware address).

#### Requirements

Firmware C2 or later.

#### Command Syntax

`:ETHERNET:CONFIG:MAC?`

#### Return String

[mac]

Variable	Description
[mac]	MAC address of the system

#### Examples

String to Send	String Returned
<code>:ETHERNET:CONFIG:MAC?</code>	<code>D0-73-7F-82-D8-01</code>

HTTP Implementation:

`http://10.10.10.10/:ETHERNET:CONFIG:MAC?`

#### See Also

[Get Static IP Address](#)  
[Get Static Subnet Mask](#)  
[Get Static Network Gateway](#)  
[Get Current Ethernet Configuration](#)

### 3.10 (r) - Get Current Ethernet Configuration

#### Description

Returns the Ethernet configuration (IP address, subnet mask and network gateway) that is currently active for the device. If DHCP is enabled this will be the settings issued dynamically by the network's DHCP server. If DHCP is disabled this will be the user configured static IP settings.

#### Requirements

Firmware C2 or later.

#### Command Syntax

```
:ETHERNET:CONFIG:LISTEN?
```

#### Return String

```
[ip];[mask];[gateway]
```

Variable	Description
[ip]	Active IP address of the device
[mask]	Subnet mask for the network
[gateway]	IP address of the network gateway

#### Examples

String to Send	String Returned
:ETHERNET:CONFIG:LISTEN?	192.100.1.1;255.255.255.0;192.100.1.0

HTTP Implementation:

```
http://10.10.10.10/:ETHERNET:CONFIG:LISTEN?
```

#### See Also

[Get Static IP Address](#)  
[Get Static Subnet Mask](#)  
[Get Static Network Gateway](#)  
[Update Ethernet Settings](#)

### 3.10 (s) - Update Ethernet Settings

#### Description

Resets the Ethernet controller so that any recently applied changes to the Ethernet configuration can be loaded. Any subsequent commands / queries to the system will need to be issued using the new Ethernet configuration.

Note: If a connection cannot be established after the INIT command has been issued it may indicate an invalid configuration was created (for example a static IP address which clashes with another device on the network). The Ethernet settings can always be overwritten by connecting to the system using the USB connection.

#### Requirements

Firmware C2 or later.

#### Command Syntax

`:ETHERNET:CONFIG:INIT`

#### Return String

`[status]`

Variable	Value	Description
<code>[status]</code>	0	Command failed
	1	Command completed successfully

#### Examples

String to Send	String Returned
<code>:ETHERNET:CONFIG:INIT</code>	1

HTTP Implementation:

`http://10.10.10.10/:ETHERNET:CONFIG:INIT`

#### See Also

[Get Current Ethernet Configuration](#)

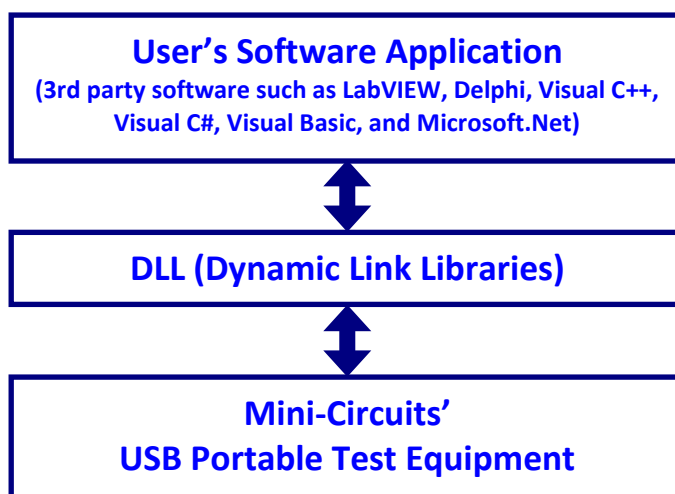
## 4 - Operating in a Windows Environment via USB

### 4.1 - The DLL (Dynamic Link Library) Concept

The Dynamic Link Library concept is Microsoft's implementation of the shared library concept in the Windows environment.

DLLs provide a mechanism for shared code and data, intended to allow a developer to distribute applications without requiring code to be re-linked or recompiled.

Mini-Circuits' software package provides DLL objects designed to allow your own application to interface with the functions of the ZTM Series test system.



*Fig 4.1-a: DLL Interface Concept*

The software package provides two DLL files, the choice of which file to use is dictated by the user's operating system:

#### 1. ActiveX com object

Designed to be used in any programming environment that supports third party ActiveX COM (Component Object Model) compliant applications.

The ActiveX file should be registered using RegSvr32 (see following sections for details).

#### 2. Microsoft.NET Class Library

A logical unit of functionality that runs under the control of the Microsoft.NET system.

## 4.1 (a) - ActiveX COM Object

ActiveX COM object DLL files are designed to be used with both 32-bit and 64-bit Windows operating systems. A 32-bit programming environment that is compatible with ActiveX is required. To develop 64-bit applications, the Microsoft.NET Class library should be used instead.

### Supported Programming Environments

Mini-Circuits' ZTM & RCM Series test systems have been tested in the following programming environments. This is not an exhaustive list and the DLL file is designed to operate in most environments that support ActiveX functionality. Please contact Mini-Circuits for support.

- Visual Studio® 6 (Visual C++ and Visual Basic)
- LabVIEW 8.0 or newer
- MATLAB 7 or newer
- Delphi
- Borland C++
- Agilent VEE
- Python

### Installation

1. Copy the DLL file to the correct directory:  
For 32-bit Windows operating systems this is C:\WINDOWS\System32  
For 64-bit Windows operating systems this is C:\WINDOWS\SysWOW64
2. Open the Command Prompt:
  - a. For Windows XP® (see *Fig 4.1-b*):
    - i. Select "All Programs" and then "Accessories" from the Start Menu
    - ii. Click on "Command Prompt" to open
  - b. For later versions of the Windows operating system you will need to have Administrator privileges in order to run the Command Prompt in "Elevated" mode (see *Fig 4.1-c* for Windows 7 and Windows 8):
    - i. Open the Start Menu/Start Screen and type "Command Prompt"
    - ii. Right-click on the shortcut for the Command Prompt
    - iii. Select "Run as Administrator"
    - iv. You may be prompted to enter the log in details for an Administrator account if the current user does not have Administrator privileges on the local PC
3. Use regsvr32 to register the DLL:  
For 32-bit Windows operating systems type (see *Fig 4.1-d*):  
`\WINDOWS\System32\Regsvr32 \WINDOWS\System32\modularzt.dll`  
For 64-bit Windows operating systems type (see *Fig 4.1-e*):  
`\WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\modularzt.dll`
4. Hit enter to confirm and a message box will appear to advise of successful registration.



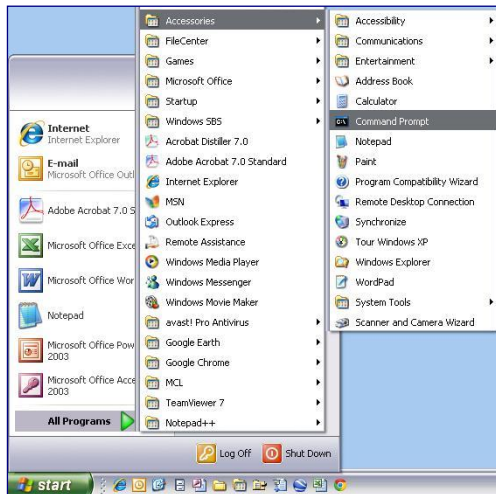


Fig 4.1-b: Opening the Command Prompt in Windows XP

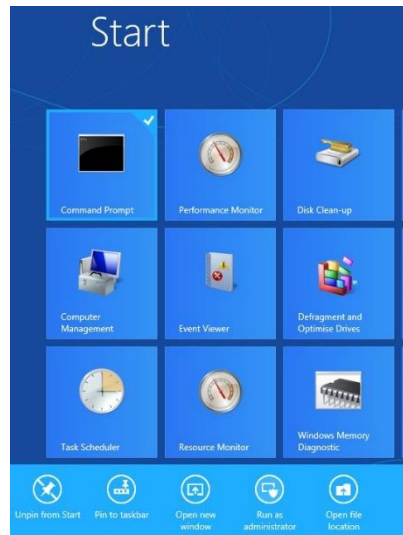
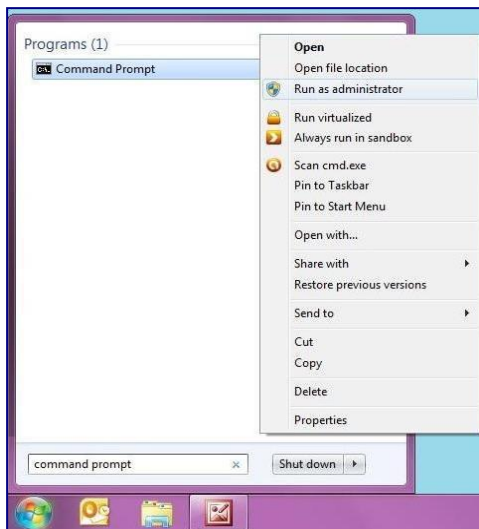


Fig 4.1-c: Opening the Command Prompt in Windows 7 (left) and Windows 8 (right)

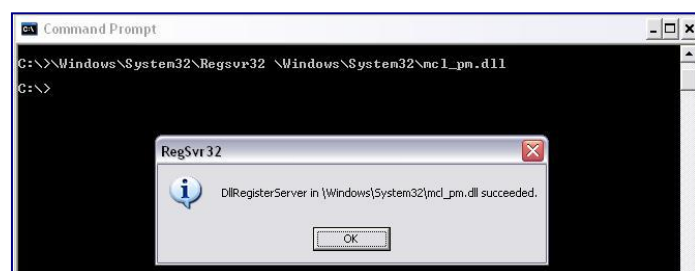


Fig 4.1-d: Registering the DLL in a 32-bit environment

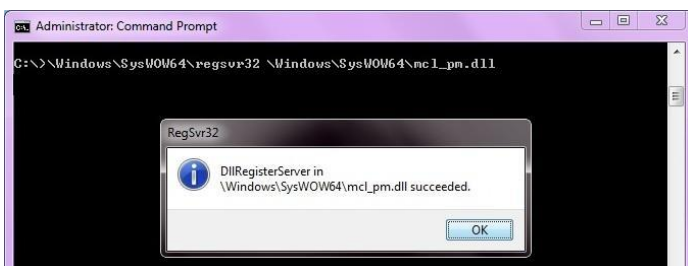


Fig 4.1-e: Registering the DLL in a 64-bit environment

#### **4.1 (b) - Microsoft.NET Class Library**

Microsoft.NET class libraries are designed to be used with both 32-bit and 64-bit Windows operating systems. To develop 64-bit applications the user must have both a 64-bit operating system and 64-bit programming environment. However, the Microsoft.NET class library is also compatible with 32-bit programming environments.

#### **Supported Programming Environments**

Mini-Circuits' ZTM & RCM Series test systems have been tested in the following programming environments. This is not an exhaustive list and the DLL file is designed to operate in most environments that support Microsoft.NET functionality. Please contact Mini-Circuits for support.

- National Instruments CVI
- Microsoft.NET (Visual C++, Visual Basic.NET, Visual C# 2003 or newer)
- LabVIEW 2009 or newer
- MATLAB 2008 or newer
- Delphi
- Borland C++

#### **Installation**

1. Copy the DLL file to the correct directory
  - a. For 32 bit Windows operating systems this is C:\WINDOWS\System32
  - b. For 64 bit Windows operating systems this is C:\WINDOWS\SysWOW64
2. **No registration is required**

## 4.2 - Referencing the DLL (Dynamic Linked Library)

In order to use the DLL functionality, some programming environments will require the user to set a reference to the relevant DLL file. Once this is done, the user just needs to declare a new instance of the USB Control class (defined within the DLL) for each modular test system to be controlled. The class is assigned to a variable which is used to call the DLL functions as needed. In the following examples, the variable names MyPTE1 and MyPTE2 have been used to represent 2 connected modular test systems.

### Example Declarations using the ActiveX DLL (modularzt.dll)

#### Visual Basic

```
Public MyPTE1 As New ModularZT.USB_Control
' Declare new ZTM / RCM Series control object, assign to MyPTE1
Public MyPTE2 As New ModularZT.USB_Control
' Declare new ZTM / RCM Series control object, assign to MyPTE2
```

#### Visual C++

```
ModularZT::USB_Control ^MyPTE1 = gcnew ModularZT::USB_Control();
// Declare new ZTM / RCM Series control object, assign to MyPTE1
ModularZT::USB_Control ^MyPTE2 = gcnew ModularZT::USB_Control();
// Declare new ZTM / RCM Series control object, assign to MyPTE2
```

#### Visual C#

```
public ModularZT.USB_Control MyPTE1 = new ModularZT.USB_Control();
// Declare new ZTM / RCM Series control object, assign to MyPTE1
public ModularZT.USB_Control MyPTE2 = new ModularZT.USB_Control();
// Declare new ZTM / RCM Series control object, assign to MyPTE2
```

#### Matlab

```
MyPTE1 = actxserver('ModularZT.USB_Control')
% Initialize new ZTM / RCM Series control object, MyPTE1
MyPTE2 = actxserver('ModularZT.USB_Control')
% Initialize new ZTM / RCM Series control object. MyPTE2
```

### Example Declarations using the .NET DLL (modularzt64.dll)

#### Visual Basic

```
Public MyPTE1 As New ModularZT64.USB_ZT
' Declare new ZTM / RCM Series control object, assign to MyPTE1
Public MyPTE2 As New ModularZT64.USB_ZT
' Declare new ZTM / RCM Series control object, assign to MyPTE2
```

#### Visual C++

```
ModularZT64.USB_ZT ^MyPTE1 = gcnew ModularZT64.USB_ZT();
// Declare new ZTM / RCM Series control object, assign to MyPTE1
ModularZT64.USB_ZT ^MyPTE2 = gcnew ModularZT64.USB_ZT();
// Declare new ZTM / RCM Series control object, assign to MyPTE2
```

#### Visual C#

```
public ModularZT64.USB_ZT MyPTE1 = new ModularZT64.USB_ZT();
// Declare new ZTM / RCM Series control object, assign to MyPTE1
public ModularZT64.USB_ZT MyPTE2 = new ModularZT64.USB_ZT();
// Declare new ZTM / RCM Series control object, assign to MyPTE2
```

#### Matlab

```
MCL_ATT=NET.addAssembly('C:\Windows\SysWOW64\ModularZT64.dll')
MyPTE1 = ModularZT64.USB_ZT % Initialize new ZTM / RCM object
MyPTE1 = ModularZT64.USB_ZT % Initialize new ZTM / RCM object
```

## 4.3 - Summary of DLL Functions

The following functions are defined in both the ActiveX and .Net DLL files. Please see the following sections for a full description of their structure and implementation.

### 4.3 (a) - USB Control Functions

- a) Short `Connect` (Optional String `SN`)
- b) Short `ConnectByAddress` (Optional Short `Address`)
- c) Void `Disconnect` ()
- d) Short `Read_ModelName` (String `ModelName`)
- e) Short `Read_SN` (String `SN`)
- f) Short `Set_Address` (Short `Address`)
- g) Short `Get_Address` ()
- h) Short `Get_Available_SN_List` (String `SN_List`)
- i) Short `Get_Available_Address_List` (String `Add_List`)
- j) Short `GetConnectionStatus` ()
- k) Short `GetUSBConnectionStatus` ()
- l) Short `Send_SCPI` (String `SndSTR`, String `RetSTR`)
- m) Short `GetExtFirmware` (Short `A0`, Short `A1`, Short `A2`, String `Firmware`)
- n) Short `GetFirmware` ()

### 4.3 (b) - Ethernet Configuration Functions

- a) Short `GetEthernet_CurrentConfig` (Int `IP1`, Int `IP2`, Int `IP3`, Int `IP4`, Int `Mask1`, Int `Mask2`, Int `Mask3`, Int `Mask4`, Int `Gateway1`, Int `Gateway2`, Int `Gateway3`, Int `Gateway4`)
- b) Short `GetEthernet_IPAddress` (Int `b1`, Int `b2`, Int `b3`, Int `b4`)
- c) Short `GetEthernet_MACAddress` (Int `MAC1`, Int `MAC2`, Int `MAC3`, Int `MAC4`, Int `MAC5`, Int `MAC6`)
- d) Short `GetEthernet_NetworkGateway` (Int `b1`, Int `b2`, Int `b3`, Int `b4`)
- e) Short `GetEthernet_SubNetMask` (Int `b1`, Int `b2`, Int `b3`, Int `b4`)
- f) Short `GetEthernet_TCPIPPort` (Int `port`)
- g) Short `GetEthernet_UseDHCP` ()
- h) Short `GetEthernet_UsePWD` ()
- i) Short `GetEthernet_PWD` (string `Pwd`)
- j) Short `SaveEthernet_IPAddress` (Int `b1`, Int `b2`, Int `b3`, Int `b4`)
- k) Short `SaveEthernet_NetworkGateway` (Int `b1`, Int `b2`, Int `b3`, Int `b4`)
- l) Short `SaveEthernet_SubnetMask` (Int `b1`, Int `b2`, Int `b3`, Int `b4`)
- m) Short `SaveEthernet_TCPIPPort` (Int `port`)
- n) Short `SaveEthernet_UseDHCP` (Int `UseDHCP`)
- o) Short `SaveEthernet_UsePWD` (Int `UsePwd`)
- p) Short `SaveEthernet_PWD` (String `Pwd`)
- q) Int `SaveEthernet_PromptMN`(Int `Enable_Prompt`)
- r) Int `GetEthernet_PromptMN`()

## 4.4 - DLL Functions for USB Control

These functions apply to all Mini-Circuits ZTM & RCM Series systems and provide a means to control the device over a USB connection.

### 4.4 (a) - Connect by Serial Number

#### Declaration

```
Short Connect(Optional String SN)
```

#### Description

Initializes the USB connection. If multiple modular test systems are connected to the same host computer, then the serial number should be included, otherwise this can be omitted. The system should be disconnected on completion of the program using the [Disconnect](#) function.

#### Parameters

Data Type	Variable	Description
String	SN	Optional. The serial number of the test system. Can be omitted if only one modular test system is connected.

#### Return Values

Data Type	Value	Description
Short	0	No connection was possible
	1	Connection successfully established
	2	Connection already established (Connect has been called more than once). The system will continue to operate normally.

#### Examples

##### Visual Basic

```
status = MyPTE1.Connect(SN)
```

##### Visual C++

```
status = MyPTE1->Connect(SN);
```

##### Visual C#

```
status = MyPTE1.Connect(SN);
```

##### Matlab

```
status = MyPTE1.Connect(SN)
```

#### See Also

[Connect by Address](#)

[Disconnect](#)

## 4.4 (b) - Connect by Address

### Declaration

**Short** **ConnectByAddress** (**Optional Short** **Address**)

### Description

This function is called to initialize the USB connection to a modular test system by referring to a user defined address. The address is an integer number from 1 to 255 which can be assigned using the [Set\\_Address](#) function (the factory default is 255). The connection process can take a few milliseconds so it is recommended that the connection be made once at the beginning of the routine and left open until the test sequence is no completed. The system should be disconnected on completion of the program using the [Disconnect](#) function.

### Parameters

Data Type	Variable	Description
<b>Short</b>	Address	Optional. The address of the system. Can be omitted if only one modular test system is connected.

### Return Values

Data Type	Value	Description
<b>Short</b>	0	No connection was possible
	1	Connection successfully established
	2	Connection already established (Connect has been called more than once)

### Examples

#### Visual Basic

```
status = MyPTE1.ConnectByAddress(5)
```

#### Visual C++

```
status = MyPTE1->ConnectByAddress(5);
```

#### Visual C#

```
status = MyPTE1.ConnectByAddress(5);
```

#### Matlab

```
status = MyPTE1.connectByAddress(5)
```

### See Also

[Connect by Serial Number](#)

[Disconnect](#)

## 4.4 (c) - Disconnect

### Declaration

```
Void Disconnect()
```

### Description

This function is called to close the connection to modular test system after completion of the test sequence. It is strongly recommended that this function is used prior to ending the program. Failure to do so may result in a connection problem with the device. Should this occur, shut down the program and unplug the system from the computer, then reconnect to start again.

### Parameters

Data Type	Variable	Description
None		

### Return Values

Data Type	Value	Description
None		

### Examples

#### Visual Basic

```
MyPTE1.Disconnect()
```

#### Visual C++

```
MyPTE1->Disconnect();
```

#### Visual C#

```
MyPTE1.Disconnect();
```

#### Matlab

```
MyPTE1.Disconnect
```

### See Also

[Connect by Serial Number](#)

[Connect by Address](#)

## 4.4 (d) - Read Model Name

### Declaration

```
Short Read_ModelName (String ModelName)
```

### Description

This function is called to determine the full Mini-Circuits part number of the connected modular test system. The user passes a string variable which is updated with the part number.

### Parameters

Data Type	Variable	Description
String	ModelName	Required. A string variable that will be updated with the Mini-Circuits part number for the modular test system.

### Return Values

Data Type	Value	Description
Short	0	Command failed
	1	Command completed successfully

### Examples

#### Visual Basic

```
If MyPTE1.Read_ModelName(ModelName) > 0 Then
    MsgBox ("The connected system is " & ModelName)
    ' Display a message stating the model name
End If
```

#### Visual C++

```
if (MyPTE1->Read_ModelName(ModelName) > 0 )
{
    MessageBox::Show("The connected system is " + ModelName);
    // Display a message stating the model name
}
```

#### Visual C#

```
if (MyPTE1.Read_ModelName(ref(ModelName)) > 0 )
{
    MessageBox.Show("The connected system is " + ModelName);
    // Display a message stating the model name
}
```

#### Matlab

```
[status, ModelName] = MyPTE1.Read_ModelName(ModelName)
if status > 0
    h = msgbox('The connected switch is ', ModelName)
    % Display a message stating the model name
end
```

### See Also

[Read Serial Number](#)

SCPI: [Get Model Name](#)



## 4.4 (e) - Read Serial Number

### Declaration

```
Short Read_SN(String SN)
```

### Description

This function is called to determine the serial number of the connected modular test system. The user passes a string variable which is updated with the serial number.

### Parameters

Data Type	Variable	Description
String	ModelName	Required. String variable that will be updated with the Mini-Circuits serial number for the test system.

### Return Values

Data Type	Value	Description
Short	0	Command failed
	1	Command completed successfully

### Examples

#### Visual Basic

```
If MyPTE1.Read_SN(SN) > 0 Then
    MsgBox ("The connected system is " & SN)
    ' Display a message stating the serial number
End If
```

#### Visual C++

```
if (MyPTE1->Read_SN(SN) > 0 )
{
    MessageBox::Show("The connected system is " + SN);
    // Display a message stating the serial number
}
```

#### Visual C#

```
if (MyPTE1.Read_SN(ref(SN)) > 0 )
{
    MessageBox.Show("The connected system is " + SN);
    // Display a message stating the serial number
}
```

#### Matlab

```
[status, SN] = MyPTE1.Read_SN(SN)
if status > 0
    h = msgbox('The connected switch is ', SN)
    % Display a message stating the serial number
end
```

### See Also

[Read Model Name](#)

SCPI: [Get Serial Number](#)

## 4.4 (f) - Set USB Address

### Declaration

```
Short Set_Address (Short Address)
```

### Description

This function allows the internal address of the connected modular test system to be changed from the factory default of 255. The system can be referred to by the address instead of the serial number (see [Connect by Address](#)).

### Parameters

Data Type	Variable	Description
Short	Address	Required. An integer value from 1 to 255

### Return Values

Data Type	Value	Description
Short	0	Command failed
	1	Command completed successfully

### Example

#### Visual Basic

```
status = MyPTE1.Set_Address(1)
```

#### Visual C++

```
status = MyPTE1->Set_Address(1);
```

#### Visual C#

```
status = MyPTE1.Set_Address(1);
```

#### Matlab

```
status = MyPTE1.Set_Address(1)
```

### See Also

[Get USB Address](#)

[Get List of Available Addresses](#)

## 4.4 (g) - Get USB Address

### Declaration

```
Short Get_Address ()
```

### Description

This function returns the address of the connected modular test system.

### Parameters

Data Type	Variable	Description
None		

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1-255	Address of the modular test system

### Examples

#### Visual Basic

```
addr = MyPTE1.Get_Address ()
```

#### Visual C++

```
addr = MyPTE1->Get_Address ();
```

#### Visual C#

```
addr = MyPTE1.Get_Address ();
```

#### Matlab

```
addr = MyPTE1.Get_Address
```

### See Also

[Set USB Address](#)

[Get List of Available Addresses](#)

## 4.4 (h) - Get List of Connected Serial Numbers

### Declaration

```
Short Get_Available_SN_List(String SN_List)
```

### Description

This function takes a user defined variable and updates it with a list of serial numbers for all available (currently connected) modular test systems.

### Parameters

Data Type	Variable	Description
String	SN_List	Required. String variable which will be updated with a list of all available serial numbers, separated by a single space character; for example "11301020001 11301020002 11301020003".

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Example

#### Visual Basic

```
If MyPTE1.Get_Available_SN_List(SN_List) > 0 Then
    array_SN() = Split(SN_List, " ")
    ' Split the list into an array of serial numbers
    For i As Integer = 0 To array_SN.Length - 1
        ' Loop through the array and use each serial number
    Next
End If
```

#### Visual C++

```
if (MyPTE1 ->Get_Available_SN_List(SN_List) > 0)
{
    // split the List into array of SN's
}
```

#### Visual C#

```
if (MyPTE1.Get_Available_SN_List(ref(SN_List)) > 0)
{
    // split the List into array of SN's
}
```

#### Matlab

```
[status, SN_List] = MyPTE1.Get_Available_SN_List(SN_List)
if status > 0
    % split the List into array of SN's
end
```

### See Also

[Connect by Serial Number](#)  
[Get List of Available Addresses](#)

## 4.4 (i) - Get List of Available Addresses

### Declaration

```
Short Get_Available_Address_List(String Add_List)
```

### Description

This function takes a user defined variable and updates it with a list of addresses of all connected modular test systems.

### Parameters

Data Type	Variable	Description
String	Add_List	Required. String variable which the function will update with a list of addresses separated by a single space character, for example, "5 101 254 255"

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Example

#### Visual Basic

```
If MyPTE1.Get_Available_Add_List(st_Ad_List) > 0 Then
    ' Get list of available addresses
    array_Ad() = Split(st_Ad_List, " ")
    ' Split the list into an array of addresses
    For i As Integer = 0 To array_Ad.Length - 1
        ' Loop through the array and use each address
    Next
End If
```

#### Visual C++

```
if (MyPTE1->Get_Available_Address_List(Add_List) > 0);
{
    // split the List into array of Addresses
}
```

#### Visual C#

```
if (MyPTE1.Get_Available_Address_List(ref(Add_List)) > 0)
{
    // split the List into array of Addresses
}
```

#### Matlab

```
[status, Add_List] = MyPTE1.Get_Available_Address_List(Add_List)
if status > 0
    % split the List into array of Addresses
end
```

### See Also

[Connect by Address](#)

[Get List of Connected Serial Numbers](#)

## 4.4 (j) - Get Software Connection Status

### Declaration

```
Short GetConnectionStatus ()
```

### Description

This function checks whether there is an open software connection to the modular test system. This will be true if the [Connect](#) function (or similar) has previously been called.

### Parameters

Data Type	Variable	Description
None		

### Return Values

Data Type	Value	Description
Short	0	No connection
Short	1	ZTM Series is connected

### Examples

#### Visual Basic

```
Status = MyPTE1.GetConnectionStatus ()
```

#### Visual C++

```
Status = MyPTE1->GetConnectionStatus ();
```

#### Visual C#

```
Status = MyPTE1.GetConnectionStatus ();
```

#### Matlab

```
Status = MyPTE1.GetConnectionStatus ()
```

### See Also

[Get USB Connection Status](#)

## 4.4 (k) - Get USB Connection Status

### Declaration

```
Short GetUSBConnectionStatus ()
```

### Description

This function checks whether the USB connection to the modular test is still active.

### Parameters

Data Type	Variable	Description
None		

### Return Values

Data Type	Value	Description
Short	0	No connection
Short	1	USB connection to modular test system is active

### Examples

#### Visual Basic

```
If MyPTE1.GetUSBConnectionStatus = 1 Then  
    ' Modular test system is connected  
End If
```

#### Visual C++

```
if (MyPTE1->GetUSBConnectionStatus() == 1)  
{  
    // Modular test system is connected  
}
```

#### Visual C#

```
if (MyPTE1.GetUSBConnectionStatus() == 1)  
{  
    // Modular test system is connected  
}
```

#### Matlab

```
usbstatus = MyPTE1.GetUSBConnectionStatus  
if usbstatus == 1  
    % Modular test system is connected  
end
```

### See Also

[Get Software Connection Status](#)

## 4.4 (I) - Send SCPI Command

### Declaration

```
Short Send_SCPI (String SndSTR, String RetSTR)
```

### Description

This function sends a SCPI command to the test system and collects the returned acknowledgement. SCPI (Standard Commands for Programmable Instruments) is a common method for communicating with and controlling instrumentation products and provides the main method for interfacing with the modular test system's internal test components.

### Parameters

Data Type	Variable	Description
String	SndSTR	Required. The SCPI command to send.
String	RetSTR	Required. User defined string which will be updated with the value returned from the test system.

### Return Values

Data Type	Value	Description
Short	0	Command failed
	1	Command completed successfully

### Examples

#### Visual Basic

```
Status = MyPTE1.Send_SCPI("RUDAT:1A:ATT:75.75", RetStr)
' Set attenuator in slot 1A to 75.75dB
```

#### Visual C++

```
Status = MyPTE1->Send_SCPI("RUDAT:1A:ATT:75.75", RetStr);
// Set attenuator in slot 1A to 75.75dB
```

#### Visual C#

```
Status = MyPTE1.Send_SCPI("RUDAT:1A:ATT:75.75", RetStr);
// Set attenuator in slot 1A to 75.75dB
```

#### Matlab

```
[Status, RetStr] = MyPTE1.Send_SCPI("RUDAT:1A:ATT:75.75", RetStr)
% Set attenuator in slot 1A to 75.75dB
```

### See Also

[SCPI Command Set for Control of ZTM-X Components](#)



## 4.4 (m) - Get Firmware

### Declaration

```
Short GetExtFirmware(Short A0, Short A1, Short A2, String Firmware)
```

### Description

This function returns the internal firmware version of the modular test system along with three reserved variables (for factory use).

### Parameters

Data Type	Variable	Description
Short	A0	Required. User defined variable for factory use only.
Short	A1	Required. User defined variable for factory use only.
Short	A2	Required. User defined variable for factory use only.
String	Firmware	Required. User defined variable which will be updated with the current firmware version, for example "B3".

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Examples

```
Visual Basic
If MyPTE1.GetExtFirmware(A0, A1, A2, Firmware) > 0 Then
    MsgBox ("Firmware version is " & Firmware)
End If

Visual C++
if (MyPTE1->GetExtFirmware(A0, A1, A2, Firmware) > 0 )
{
    MessageBox::Show("Firmware version is " + Firmware);
}

Visual C#
if (MyPTE1.GetExtFirmware(ref(A0, A1, A2, Firmware)) > 0 )
{
    MessageBox.Show("Firmware version is " + Firmware);
}

Matlab
[status, A0, A1, A2, Firmware] = MyPTE1.GetExtFirmware(A0, A1, A2, Firmware)
if status > 0
    h = msgbox('Firmware version is ', Firmware)
end
```

### See Also

SCPI: [Get Firmware](#)

## 4.4 (n) - Get Firmware Version (Antiquated)

### Declaration

```
Short GetFirmware()
```

### Description

This function is antiquated, [GetExtFirmware](#) should be used instead. The function returns a numeric value corresponding to the internal firmware version of the test system.

### Parameters

Data Type	Variable	Description
None		

### Return Values

Data Type	Value	Description
Short	Firmware	Version number of the firmware

### Examples

#### Visual Basic

```
FW = MyPTE1.GetFirmware()
```

#### Visual C++

```
FW = MyPTE1->GetFirmware();
```

#### Visual C#

```
FW = MyPTE1.GetFirmware();
```

#### Matlab

```
FW = MyPTE1.GetFirmware()
```

### See Also

[Get Firmware](#)

## 4.5 - DLL Functions for Ethernet Configuration

These functions provide a means for identifying and configuring the Ethernet settings such as IP address, TCP/IP port and network gateway. They can only be called while the system is connected via the USB interface. In order to determine the current connection status (for example the IP address of the network gateway) the test system must also be connected to the network via the RJ45 port.

### 4.5 (a) - Get Ethernet Configuration

#### Declaration

```
Short GetEthernet_CurrentConfig(Int IP1, Int IP2, Int IP3, Int IP4,
                                Int Mask1, Int Mask2, Int Mask3, Int Mask4,
                                Int Gateway1, Int Gateway2, Int Gateway3, Int Gateway4)
```

#### Description

This function returns the current IP configuration of the connected modular test system in a series of user defined variables. The settings checked are IP address, subnet mask and network gateway.

#### Parameters

Data Type	Variable	Description
Int	IP1	Required. Integer variable which will be updated with the first (highest order) octet of the IP address.
Int	IP2	Required. Integer variable which will be updated with the second octet of the IP address.
Int	IP3	Required. Integer variable which will be updated with the third octet of the IP address.
Int	IP4	Required. Integer variable which will be updated with the last (lowest order) octet of the IP address.
Int	Mask1	Required. Integer variable which will be updated with the first (highest order) octet of the subnet mask.
Int	Mask2	Required. Integer variable which will be updated with the second octet of the subnet mask.
Int	Mask3	Required. Integer variable which will be updated with the third octet of the subnet mask.
Int	Mask4	Required. Integer variable which will be updated with the last (lowest order) octet of the subnet mask.
Int	Gateway1	Required. Integer variable which will be updated with the first (highest order) octet of the network gateway.
Int	Gateway2	Required. Integer variable which will be updated with the second octet of the network gateway.
Int	Gateway3	Required. Integer variable which will be updated with the third octet of the network gateway.
Int	Gateway4	Required. Integer variable which will be updated with the last (lowest order) octet of the network gateway.

## Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

## Example

### Visual Basic

```
If MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4, M1, M2, M3, M4,
    _ GW1, GW2, GW3, GW4) > 0 Then

    MsgBox ("IP address: " & IP1 & "." & IP2 & "." & IP3 & "." & IP4)
    MsgBox ("Subnet Mask: " & M1 & "." & M2 & "." & M3 & "." & M4)
    MsgBox ("Gateway: " & GW1 & "." & GW2 & "." & GW3 & "." & GW4)

End If
```

### Visual C++

```
if (MyPTE1->GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4, M1, M2, M3, M4,
    _ GW1, GW2, GW3, GW4) > 0)
{
    MessageBox::Show("IP address: " + IP1 + "." + IP2 + "." + IP3 + "." +
        _ + IP4);
    MessageBox::Show("Subnet Mask: " + M1 + "." + M2 + "." + M3 + "." +
        _ M4);
    MessageBox::Show("Gateway: " + GW1 + "." + GW2 + "." + GW3 + "." +
        _ GW4);
}
```

### Visual C#

```
if (MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4, M1, M2, M3, M4,
    _ GW1, GW2, GW3, GW4) > 0)
{
    MessageBox.Show("IP address: " + IP1 + "." + IP2 + "." + IP3 + "." +
        _ + IP4);
    MessageBox.Show("Subnet Mask: " + M1 + "." + M2 + "." + M3 + "." +
        _ M4);
    MessageBox.Show("Gateway: " + GW1 + "." + GW2 + "." + GW3 + "." +
        _ GW4);
}
```

### Matlab

```
[status, IP1, IP2, IP3, IP4, M1, M2, M3, M4, GW1, GW2, GW3, GW4] =
MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4, M1, M2, M3, M4, GW1,
GW2, GW3, GW4)
if status > 0
    h = msgbox ("IP address: ", IP1, ".", IP2, ".", IP3, ".", IP4)
    h = msgbox ("Subnet Mask: ", M1, "." & M2, "." & M3, ".", M4)
    h = msgbox ("Gateway: ", GW1, ".", GW2, ".", GW3, ".", GW4)
end
```

## See Also

[Get MAC Address](#)

[Get TCP/IP Port](#)

## 4.5 (b) - Get IP Address

### Declaration

```
Short GetEthernet_IPAddress(Int b1, Int b2, Int b3, Int b4)
```

### Description

This function returns the current IP address of the connected system in a series of user defined variables (one per octet).

### Parameters

Data Type	Variable	Description
Int	IP1	Required. Integer variable which will be updated with the first (highest order) octet of the IP address (for example "192" for the IP address "192.168.1.0").
Int	IP2	Required. Integer variable which will be updated with the second octet of the IP address (for example "168" for the IP address "192.168.1.0").
Int	IP3	Required. Integer variable which will be updated with the third octet of the IP address (for example "1" for the IP address "192.168.1.0").
Int	IP4	Required. Integer variable which will be updated with the last (lowest order) octet of the IP address (for example "0" for the IP address "192.168.1.0").

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

## Example

### Visual Basic

```
If MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4) > 0 Then
    MsgBox ("IP address: " & IP1 & "." & IP2 & "." & IP3 & "." & IP4)
End If
```

### Visual C++

```
if (MyPTE1->GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4) > 0)
{
    MessageBox::Show("IP address: " + IP1 + "." + IP2 + "." + IP3 + "."
        + IP4);
}
```

### Visual C#

```
if (MyPTE1.GetEthernet_CurrentConfig(IP1, IP2, IP3, IP4) > 0)
{
    MessageBox.Show("IP address: " + IP1 + "." + IP2 + "." + IP3 + "."
        + IP4);
}
```

### Matlab

```
[status, IP1, IP2, IP3, IP4] = MyPTE1.GetEthernet_CurrentConfig(IP1, IP2,
IP3, IP4)
if status > 0
    h = msgbox ("IP address: ", IP1, ".", IP2, ".", IP3, ".", IP4)
end
```

## See Also

[Get Ethernet Configuration](#)

[Get TCP/IP Port](#)

[Save IP Address](#)

[Save TCP/IP Port](#)

## 4.5 (c) - Get MAC Address

### Declaration

```
Short GetEthernet_MACAddress (Int MAC1, Int MAC2, Int MAC3, Int MAC4,  
                              Int MAC5, Int MAC6)
```

### Description

This function returns the MAC (media access control) address, the physical address, of the connected system as a series of decimal values (one for each of the 6 numeric groups).

### Parameters

Data Type	Variable	Description
Int	MAC1	Required. Integer variable which will be updated with the decimal value of the first numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC1=11
Int	MAC2	Required. Integer variable which will be updated with the decimal value of the second numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC2=47
Int	MAC3	Required. Integer variable which will be updated with the decimal value of the third numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC3=165
Int	MAC4	Required. Integer variable which will be updated with the decimal value of the fourth numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC4=103
Int	MAC5	Required. Integer variable which will be updated with the decimal value of the fifth numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC5=137
Int	MAC6	Required. Integer variable which will be updated with the decimal value of the last numeric group of the MAC address. For example: MAC address =11:47:165:103:137:171 MAC6=171

## Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

## Example

### Visual Basic

```
If MyPTE1.GetEthernet_MACAddress(M1, M2, M3, M4, M5, M6) > 0 Then
    MsgBox ("MAC address: " & M1 & ":" & M2 & ":" & M3 & ":" & M4 & ":" & M5 & ":" & M6)
End If
```

### Visual C++

```
if (MyPTE1->GetEthernet_MACAddress(M1, M2, M3, M4, M5, M6) > 0)
{
    MessageBox::Show("MAC address: " + M1 + "." + M2 + "." + M3 + "." + M4 + "." + M5 + "." + M6);
}
```

### Visual C#

```
if (MyPTE1.GetEthernet_MACAddress(M1, M2, M3, M4, M5, M6) > 0)
{
    MessageBox.Show("MAC address: " + M1 + "." + M2 + "." + M3 + "." + M4 + "." + M5 + "." + M6);
}
```

### Matlab

```
[status, M1, M2, M3, M4, M5, M6] = MyPTE1.GetEthernet_MACAddress(M1, M2, M3, M4, M5, M6)
if status > 0
    h=msgbox("MAC address: ", M1, ".", M2, ".", M3, ".", M4, ".", M5, ".", M6)
end
```

## See Also

[Get Ethernet Configuration](#)



## 4.5 (d) - Get Network Gateway

### Declaration

```
Short GetEthernet_NetworkGateway(Int b1, Int b2, Int b3, Int b4)
```

### Description

This function returns the IP address of the network gateway to which the system is currently connected. A series of user defined variables are passed to the function to be updated with the IP address (one per octet).

### Parameters

Data Type	Variable	Description
Int	IP1	Required. Integer variable which will be updated with the first (highest order) octet of the IP address (for example "192" for the IP address "192.168.1.0").
Int	IP2	Required. Integer variable which will be updated with the second octet of the IP address (for example "168" for the IP address "192.168.1.0").
Int	IP3	Required. Integer variable which will be updated with the third octet of the IP address (for example "1" for the IP address "192.168.1.0").
Int	IP4	Required. Integer variable which will be updated with the last (lowest order) octet of the IP address (for example "0" for the IP address "192.168.1.0").

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

## Example

### Visual Basic

```
If MyPTE1.GetEthernet_NetworkGateway(IP1, IP2, IP3, IP4) > 0 Then  
    MsgBox ("Gateway: " & IP1 & "." & IP2 & "." & IP3 & "." & IP4)  
End If
```

### Visual C++

```
if (MyPTE1->GetEthernet_NetworkGateway(IP1, IP2, IP3, IP4) > 0)  
{  
    MessageBox::Show("Gateway: " + IP1 + "." + IP2 + "." + IP3 + "."  
                      + IP4);  
}
```

### Visual C#

```
if (MyPTE1.GetEthernet_NetworkGateway(IP1, IP2, IP3, IP4) > 0)  
{  
    MessageBox.Show("Gateway: " + IP1 + "." + IP2 + "." + IP3 + "."  
                    + IP4);  
}
```

### Matlab

```
[status, IP1, IP2, IP3, IP4] = MyPTE1.GetEthernet_NetworkGateway(IP1, IP2,  
IP3, IP4)  
if status > 0  
    h = msgbox ("Gateway: ", IP1, ".", IP2, ".", IP3, ".", IP4)  
end
```

## See Also

[Get Ethernet Configuration](#)

[Save Network Gateway](#)

## 4.5 (e) - Get Subnet Mask

### Declaration

```
Short GetEthernet_SubNetMask(Int b1, Int b2, Int b3, Int b4)
```

### Description

This function returns the subnet mask used by the network gateway to which the system is currently connected. A series of user defined variables are passed to the function to be updated with the subnet mask (one per octet).

### Parameters

Data Type	Variable	Description
Int	b1	Required. Integer variable which will be updated with the first (highest order) octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
Int	b2	Required. Integer variable which will be updated with the second octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
Int	b3	Required. Integer variable which will be updated with the third octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
Int	b4	Required. Integer variable which will be updated with the last (lowest order) octet of the subnet mask (for example "0" for the subnet mask "255.255.255.0").

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

## Example

### Visual Basic

```
If MyPTE1.GetEthernet_SubNetMask(b1, b2, b3, b4) > 0 Then
    MsgBox ("Subnet mask: " & b1 & "." & b2 & "." & b3 & "." & b4)
End If
```

### Visual C++

```
if (MyPTE1->GetEthernet_SubNetMask(b1, b2, b3, b4) > 0)
{
    MessageBox::Show("Subnet mask: " + b1 + "." + b2 + "." + b3 + "."
                    + b4);
}
```

### Visual C#

```
if (MyPTE1.GetEthernet_SubNetMask(b1, b2, b3, b4) > 0)
{
    MessageBox.Show("Subnet mask: " + b1 + "." + b2 + "." + b3 + "."
                    + b4);
}
```

### Matlab

```
[status, b1, b2, b3, b4] = MyPTE1.GetEthernet_SubNetMask(b1, b2, b3, b4)
if status > 0
    h = msgbox ("Subnet mask: ", b1, ".", b2, ".", b3, ".", b4)
end
```

## See Also

[Get Ethernet Configuration](#)

[Save Subnet Mask](#)

## 4.5 (f) - Get TCP/IP Port

### Declaration

```
Short GetEthernet_TCPIPPort(Int port)
```

### Description

This function returns the TCP/IP port used by the test system for HTTP communication. The default is port 80.

Note: Port 23 is reserved for Telnet communication and cannot be set as the HTTP port.

### Parameters

Data Type	Variable	Description
Int	port	Required. Integer variable which will be updated with the TCP/IP port.

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Example

#### Visual Basic

```
If MyPTE1.GetEthernet_SubNetMask(port) > 0 Then
    MsgBox ("Port: " & port)
End If
```

#### Visual C++

```
if (MyPTE1->GetEthernet_SubNetMask(port) > 0)
{
    MessageBox::Show("Port: " + port);
}
```

#### Visual C#

```
if (MyPTE1.GetEthernet_SubNetMask(port) > 0)
{
    MessageBox.Show("Port: " + port);
}
```

#### Matlab

```
[status, port] = MyPTE1.GetEthernet_SubNetMask(port)
if status > 0
    h = msgbox ("Port: ", port)
end
```

### See Also

[Get Ethernet Configuration](#)  
[Save TCP/IP Port](#)

## 4.5 (g) - Get DHCP Status

### Declaration

```
Short GetEthernet_UseDHCP ()
```

### Description

This function indicates whether the test system is using DHCP (dynamic host control protocol), in which case the IP configuration is derived from a network server; or user defined “static” IP settings.

### Parameters

Data Type	Variable	Description
None		

### Return Values

Data Type	Value	Description
Short	0	DHCP not in use (IP settings are static and manually configured)
Short	1	DHCP in use (IP settings are assigned automatically by the network)

### Example

#### Visual Basic

```
DHCPstatus = MyPTE1.GetEthernet_UseDHCP ()
```

#### Visual C++

```
DHCPstatus = MyPTE1->GetEthernet_UseDHCP ();
```

#### Visual C#

```
DHCPstatus = MyPTE1.GetEthernet_UseDHCP ();
```

#### Matlab

```
DHCPstatus = MyPTE1.GetEthernet_UseDHCP
```

### See Also

[Get Ethernet Configuration](#)

[Use DHCP](#)

## 4.5 (h) - Get Password Status

### Declaration

```
Short GetEthernet_UsePWD ()
```

### Description

This function indicates whether the modular test system is currently configured to require a password for HTTP/Telnet communication.

### Parameters

Data Type	Variable	Description
None		

### Return Values

Data Type	Value	Description
Short	0	Password not required
Short	1	Password required

### Example

#### Visual Basic

```
PWDstatus = MyPTE1.GetEthernet_UsePWD ()
```

#### Visual C++

```
PWDstatus = MyPTE1->GetEthernet_UsePWD ();
```

#### Visual C#

```
PWDstatus = MyPTE1.GetEthernet_UsePWD ();
```

#### Matlab

```
PWDstatus = MyPTE1.GetEthernet_UsePWD
```

### See Also

[Get Password](#)

[Use Password](#)

[Set Password](#)

## 4.5 (i) - Get Password

### Declaration

```
Short GetEthernet_PWD (String Pwd)
```

### Description

This function returns the current password used by the modular test system for HTTP/Telnet communication. The password will be returned even if the device is not currently configured to require a password.

### Parameters

Data Type	Variable	Description
String	Pwd	Required. String variable which will be updated with the password.

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Example

#### Visual Basic

```
If MyPTE1.GetEthernet_PWD(pwd) > 0 Then
    MsgBox ("Password: " & pwd)
End If
```

#### Visual C++

```
if (MyPTE1->GetEthernet_PWD(pwd) > 0)
{
    MessageBox::Show("Password: " + pwd);
}
```

#### Visual C#

```
if (MyPTE1.GetEthernet_PWD(pwd) > 0)
{
    MessageBox.Show("Password: " + pwd);
}
```

#### Matlab

```
[status, pwd] = MyPTE1.GetEthernet_PWD(pwd)
if status > 0
    h = msgbox ("Password: ", pwd)
end
```

### See Also

[Get Password Status](#)

[Use Password](#)

[Set Password](#)



## 4.5 (j) - Save IP Address

### Declaration

```
Short SaveEthernet_IPAddress(Int b1, Int b2, Int b3, Int b4)
```

### Description

This function sets a static IP address to be used by the connected test system.

Note: this could subsequently be overwritten automatically if DHCP is enabled (see [Use DHCP](#)).

### Parameters

Data Type	Variable	Description
Int	IP1	Required. First (highest order) octet of the IP address to set (for example "192" for the IP address "192.168.1.0").
Int	IP2	Required. Second octet of the IP address to set (for example "168" for the IP address "192.168.1.0").
Int	IP3	Required. Third octet of the IP address to set (for example "1" for the IP address "192.168.1.0").
Int	IP4	Required. Last (lowest order) octet of the IP address to set (for example "0" for the IP address "192.168.1.0").

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Example

#### Visual Basic

```
status = MyPTE1.SaveEthernet_IPAddress(192, 168, 1, 0)
```

#### Visual C++

```
status = MyPTE1->SaveEthernet_IPAddress(192, 168, 1, 0);
```

#### Visual C#

```
status = MyPTE1.SaveEthernet_IPAddress(192, 168, 1, 0);
```

#### Matlab

```
status = MyPTE1.SaveEthernet_IPAddress(192, 168, 1, 0)
```

### See Also

[Get Ethernet Configuration](#)

[Get IP Address](#)

## 4.5 (k) - Save Network Gateway

### Declaration

```
Short SaveEthernet_NetworkGateway(Int b1, Int b2, Int b3, Int b4)
```

### Description

This function sets the IP address of the network gateway to which the system should connect.

Note: this could subsequently be overwritten automatically if DHCP is enabled (see [Use DHCP](#)).

### Parameters

Data Type	Variable	Description
Int	IP1	Required. First (highest order) octet of the network gateway IP address (for example "192" for the IP address "192.168.1.0").
Int	IP2	Required. Second octet of the network gateway IP address (for example "168" for the IP address "192.168.1.0").
Int	IP2	Required. Third octet of the network gateway IP address (for example "1" for the IP address "192.168.1.0").
Int	IP4	Required. Last (lowest order) octet of the network gateway IP address (for example "0" for the IP address "192.168.1.0").

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Example

#### Visual Basic

```
status = MyPTE1.SaveEthernet_NetworkGateway(192, 168, 1, 0)
```

#### Visual C++

```
status = MyPTE1->SaveEthernet_NetworkGateway(192, 168, 1, 0);
```

#### Visual C#

```
status = MyPTE1.SaveEthernet_NetworkGateway(192, 168, 1, 0);
```

#### Matlab

```
status = MyPTE1.SaveEthernet_NetworkGateway(192, 168, 1, 0)
```

### See Also

[Get Ethernet Configuration](#)

[Get Network Gateway](#)

## 4.5 (I) - Save Subnet Mask

### Declaration

```
Short SaveEthernet_SubnetMask(Int b1, Int b2, Int b3, Int b4)
```

### Description

This function sets the subnet mask of the network to which the system should connect.

Note: this could subsequently be overwritten automatically if DHCP is enabled (see [Use DHCP](#)).

### Parameters

Data Type	Variable	Description
Int	IP1	Required. First (highest order) octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
Int	IP2	Required. Second octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
Int	IP3	Required. Third octet of the subnet mask (for example "255" for the subnet mask "255.255.255.0").
Int	IP4	Required. Last (lowest order) octet of the subnet mask (for example "0" for the subnet mask "255.255.255.0").

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Example

#### Visual Basic

```
status = MyPTE1.SaveEthernet_SubnetMask(255, 255, 255, 0)
```

#### Visual C++

```
status = MyPTE1->SaveEthernet_SubnetMask(255, 255, 255, 0);
```

#### Visual C#

```
status = MyPTE1.SaveEthernet_SubnetMask(255, 255, 255, 0);
```

#### Matlab

```
status = MyPTE1.SaveEthernet_SubnetMask(255, 255, 255, 0)
```

### See Also

[Get Ethernet Configuration](#)

[Get Subnet Mask](#)

## 4.5 (m) - Save TCP/IP Port

### Declaration

```
Short SaveEthernet_TCPIPPort(Int port)
```

### Description

This function sets the TCP/IP port used by the system for HTTP communication. The default is port 80.

Note: Port 23 is reserved for Telnet communication and cannot be set as the HTTP port.

### Parameters

Data Type	Variable	Description
Int	port	Required. Numeric value of the TCP/IP port.

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Example

#### Visual Basic

```
status = MyPTE1.SaveEthernet_TCPIPPort(70)
```

#### Visual C++

```
status = MyPTE1->SaveEthernet_TCPIPPort(70);
```

#### Visual C#

```
status = MyPTE1.SaveEthernet_TCPIPPort(70);
```

#### Matlab

```
status = MyPTE1.SaveEthernet_TCPIPPort(70)
```

### See Also

[Get TCP/IP Port](#)

## 4.5 (n) - Use DHCP

### Declaration

```
Short SaveEthernet_UseDHCP (Int UseDHCP)
```

### Description

This function enables or disables DHCP (dynamic host control protocol). When enabled the IP configuration of the system is assigned automatically by the network server; when disabled the user defined “static” IP settings apply.

### Parameters

Data Type	Variable	Description
Int	UseDHCP	Required. Integer value to set the DHCP mode: 0 - DHCP disabled (static IP settings used) 1 - DHCP enabled (IP setting assigned by network)

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Example

```

Visual Basic
    status = MyPTE1.SaveEthernet_UseDHCP(1)
Visual C++
    status = MyPTE1->SaveEthernet_UseDHCP(1);
Visual C#
    status = MyPTE1.SaveEthernet_UseDHCP(1);
Matlab
    status = MyPTE1.SaveEthernet_UseDHCP(1)
  
```

### See Also

[Get DHCP Status](#)

## 4.5 (o) - Use Password

### Declaration

```
Short SaveEthernet_UsePWD (Int UsePwd)
```

### Description

This function enables or disables the password requirement for HTTP/Telnet communication with the system.

### Parameters

Data Type	Variable	Description
Int	UseDHCP	Required. Integer value to set the password mode: 0 – Password not required 1 – Password required

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Example

#### Visual Basic

```
status = MyPTE1.SaveEthernet_UsePWD(1)
```

#### Visual C++

```
status = MyPTE1->SaveEthernet_UsePWD(1);
```

#### Visual C#

```
status = MyPTE1.SaveEthernet_UsePWD(1);
```

#### Matlab

```
status = MyPTE1.SaveEthernet_UsePWD(1)
```

### See Also

[Get Password Status](#)

[Get Password](#)

[Set Password](#)

## 4.5 (p) - Set Password

### Declaration

```
Short SaveEthernet_PWD (String Pwd)
```

### Description

This function sets the password used by the system for HTTP/Telnet communication. The password will not affect switch operation unless [Use Password](#) is also enabled.

### Parameters

Data Type	Variable	Description
String	Pwd	Required. The password to set (20 characters maximum).

### Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

### Example

#### Visual Basic

```
status = MyPTE1.SaveEthernet_PWD("123")
```

#### Visual C++

```
status = MyPTE1->SaveEthernet_PWD("123");
```

#### Visual C#

```
status = MyPTE1.SaveEthernet_PWD("123");
```

#### Matlab

```
status = MyPTE1.SaveEthernet_PWD("123")
```

### See Also

[Get Password Status](#)

[Get Password](#)

[Use Password](#)

## 4.5 (q) - Set Telnet Prompt

### Declaration

```
Int SaveEthernet_PromptMN(Int Enable_Prompt)
```

### Description

Determines the prompt to be returned by the test system for Telnet communication. By default the prompt is disabled so the response for Telnet communication is a new line character. When enabled, a full prompt is returned to the unit in response to all Telnet communication, taking the form "MODEL\_NAME>".

### Parameters

Data Type	Variable	Description
Int	Enable_Prompt	0 = Disabled (new line character returned) 1 = Enabled (full model name prompt returned)

### Return Values

Data Type	Value	Description
Int	0	Command failed
	1	Command completed successfully

### Example

```

Visual Basic
    status = MyPTE1.SaveEthernet_PromptMN(1)
Visual C++
    status = MyPTE1->SaveEthernet_PromptMN(1);
Visual C#
    status = MyPTE1.SaveEthernet_PromptMN(1);
Matlab
    status = MyPTE1.SaveEthernet_PromptMN(1)

```

### See Also

[Get Telnet Prompt Status](#)



## 4.5 (r) - Get Telnet Prompt Status

### Declaration

```
Int GetEthernet_PromptMN()
```

### Description

Indicates whether a full prompt is to be returned by the test system for Telnet communication. By default the prompt is disabled so the response for Telnet communication is a new line character. When enabled, a full prompt is returned to the unit in response to all Telnet communication, taking the form "MODEL\_NAME>".

### Return Values

Data Type	Value	Description
Int	0	Disabled (new line character returned)
	1	Enabled (full model name prompt returned)

### Example

#### Visual Basic

```
status = MyPTE1.GetEthernet_PromptMN()
```

#### Visual C++

```
status = MyPTE1->GetEthernet_PromptMN();
```

#### Visual C#

```
status = MyPTE1.GetEthernet_PromptMN();
```

#### Matlab

```
status = MyPTE1.GetEthernet_PromptMN()
```

### See Also

[Set Telnet Prompt](#)

## 5 - Operating in a Linux Environment via USB

To open a USB connection to Mini-Circuits ZTM & RCM Series modular test systems, the Vendor ID and Product ID are required:

- Mini-Circuits Vendor ID: 0x20CE
- ZTM Series Product ID: 0x22

Communication with the test system is carried out by way of USB Interrupt. The transmitted and received buffer sizes are 64 Bytes each:

- Transmit Array = [Byte 0][Byte1][Byte2]...[Byte 63]
- Returned Array = [Byte 0][Byte1][Byte2]...[Byte 63]

In most cases, the full 64 byte buffer size is not needed so any unused bytes become “don’t care” bytes; they can take on any value without affecting the operation of the system.

### 5.1 - Summary of Commands

The commands that can be sent to the ZTM Series are summarized in the table below and detailed on the following pages.

	Description	Command Code (Byte 0)	Comments
<b>a</b>	Get Device Model Name	40	
<b>b</b>	Get Device Serial Number	41	
<b>c</b>	Send SCPI Command	1 or 2 or 42	
<b>d</b>	Get Firmware	99	
<b>e</b>	Get Internal Temperature	114 115 118	Sensor 1 Sensor 2 Sensor 3

## 5.2 - Detailed Description of Commands

### 5.2 (a) - Get Device Model Name

#### Description

Returns the Mini-Circuits part number of the connected modular test system.

#### Transmit Array

Byte	Data	Description
0	40	Interrupt code for Get Device Model Name
1- 63	Not significant	"Don't care" bytes, can be any value

#### Returned Array

Byte	Data	Description
0	40	Interrupt code for Get Device Model Name
1 to (n-1)	Model Name	Series of bytes containing the ASCII code for each character in the model name
n	0	Zero value byte to indicate the end of the model name
(n+1) to 63	Not significant	"Don't care" bytes, can be any value

#### Example

The following array would be returned for ZTM-999 (see [Appendix A](#) for conversions between decimal, binary and ASCII characters):

Byte	Data	Description
0	40	Interrupt code for Get Device Model Name
1	90	ASCII character code for Z
2	84	ASCII character code for T
3	77	ASCII character code for M
4	45	ASCII character code for -
5	57	ASCII character code for 9
6	57	ASCII character code for 9
7	57	ASCII character code for 9
8	0	Zero value byte to indicate end of string

#### See Also

[Get Device Serial Number](#)  
 SCPI: [Get Model Name](#)

## 5.2 (b) - Get Device Serial Number

### Description

Returns the serial number of the connected modular test system.

### Transmit Array

Byte	Data	Description
0	41	Interrupt code for Get Device Serial Number
1 - 63	Not significant	"Don't care" bytes, can be any value

### Returned Array

Byte	Data	Description
0	41	Interrupt code for Get Device Serial Number
1 to (n-1)	Serial Number	Series of bytes containing the ASCII code for each character in the serial number
n	0	Zero value byte to indicate the end of the serial number
(n+1) to 63	Not significant	"Don't care" bytes, can be any value

### Example

The following example indicates that the connected ZTM Series system has serial number 1130922011 (see [Appendix A](#) for conversions between decimal, binary and ASCII characters):

Byte	Data	Description
0	41	Interrupt code for Get Device Serial Number
1	49	ASCII character code for 1
2	49	ASCII character code for 1
3	51	ASCII character code for 3
4	48	ASCII character code for 0
5	57	ASCII character code for 9
6	50	ASCII character code for 2
7	50	ASCII character code for 2
8	48	ASCII character code for 0
9	49	ASCII character code for 1
10	49	ASCII character code for 1
11	0	Zero value byte to indicate end of string

### See Also

[Get Device Model Name](#)  
 SCPI: [Get Serial Number](#)

## 5.2 (c) - Send SCPI Command

### Description

This function sends an SCPI command to the modular test system and collects the returned acknowledgement. SCPI (Standard Commands for Programmable Instruments) is a common method for communicating with and controlling instrumentation products and provides the main method for interfacing with the ZTM Series system's internal test components.

### Transmit Array

Byte	Data	Description
0	1 or 2 or 42	Interrupt code for Send SCPI Command Note: Any of 1, 2 or 42 can be used as the command byte and the same value will be received in byte 0 of the returned array. 42 can be convenient in some environments since it can easily be represented by its character value of "**".
1 - 63	SCPI Transmit String	The SCPI command to send represented as a series of ASCII character codes, one character code per byte

### Returned Array

Byte	Data	Description
0	1 or 2 or 42	Interrupt code for Send SCPI Command
1 to (n-1)	SCPI Return String	The SCPI return string, one character per byte, represented as ASCII character codes
n	0	Zero value byte to indicate the end of the SCPI return string
(n+1) to 63	Not significant	"Don't care" bytes, can be any value

### Example 1 (Get Model Name)

The SCPI command to request the model name is `:MN?` (see [Get Model Name](#))

The ASCII character codes representing the 4 characters in this command should be sent in bytes 1 to 4 of the transmit array as follows (see [Appendix A](#) for conversions between decimal, binary and ASCII characters):

Byte	Data	Description
0	1	Interrupt code for Send SCPI Command
1	58	ASCII character code for :
2	77	ASCII character code for M
3	78	ASCII character code for N
4	63	ASCII character code for ?

The returned array for ZTM-999 would be as follows:

Byte	Data	Description
0	1	Interrupt code for Send SCPI Command
1	90	ASCII character code for Z
2	84	ASCII character code for T
3	77	ASCII character code for M
4	45	ASCII character code for -
5	57	ASCII character code for 9
6	57	ASCII character code for 9
7	57	ASCII character code for 9
8	0	Zero value byte to indicate end of string

## Example 2 (Set Attenuator)

The SCPI command to set an attenuator in slot 1A to 70.25dB is `:RUDAT:1A:ATT:70.25` (see [Set Attenuation](#)). The ASCII character codes representing the 19 characters in this command should be sent in bytes 1 to 19 of the transmit array as follows:

Byte	Data	Description
0	1	Interrupt code for Send SCPI Command
1	58	ASCII character code for :
2	82	ASCII character code for R
3	85	ASCII character code for U
4	68	ASCII character code for D
5	65	ASCII character code for A
6	84	ASCII character code for T
7	58	ASCII character code for :
8	49	ASCII character code for 1
9	65	ASCII character code for A
10	58	ASCII character code for :
11	65	ASCII character code for A
12	84	ASCII character code for T
13	84	ASCII character code for T
14	58	ASCII character code for :
15	55	ASCII character code for 7
16	48	ASCII character code for 0
17	46	ASCII character code for .
18	50	ASCII character code for 2
19	53	ASCII character code for 5

The returned array to indicate success would be:

Byte	Data	Description
0	1	Interrupt code for Send SCPI Command
1	49	ASCII character code for 1
2	32	ASCII character code for space character
3	45	ASCII character code for -
4	32	ASCII character code for space character
5	83	ASCII character code for S
6	85	ASCII character code for U
7	67	ASCII character code for C
8	67	ASCII character code for C
9	69	ASCII character code for E
10	83	ASCII character code for S
11	83	ASCII character code for S
12	0	Zero value byte to indicate end of string

See Also

[SCPI Command Set for Control of ZTM-X Components](#)

## 5.2 (d) - Get Firmware

### Description

This function returns the internal firmware version of the modular test system.

### Transmit Array

Byte	Data	Description
0	99	Interrupt code for Get Firmware
1- 63	Not significant	"Don't care" bytes, can be any value

### Returned Array

Byte	Data	Description
0	99	Interrupt code for Get Firmware
1	Reserved	Internal code for factory use only
2	Reserved	Internal code for factory use only
3	Reserved	Internal code for factory use only
4	Reserved	Internal code for factory use only
5	Firmware Letter	ASCII code for the first character in the firmware revision identifier
6	Firmware Number	ASCII code for the second character in the firmware revision identifier
7-63	Not significant	"Don't care" bytes, could be any value

### Example

The below returned array indicates that the system has firmware version "C3" (see [Appendix A](#) for conversions between decimal, binary and ASCII characters):

Byte	Data	Description
0	99	Interrupt code for Get Firmware
1	49	Not significant
2	77	Not significant
3	78	Not significant
4	63	Not significant
5	67	ASCII character code for C
6	51	ASCII character code for 3

### See Also

SCPI: [Get Firmware](#)



## 5.2 (e) - Get Internal Temperature

### Description

Returns the internal temperature of the modular test system in degrees Celsius to 2 decimal places.

### Transmit Array

Byte	Data	Description
0	114, 115 or 118	Interrupt code for Get Internal Temperature: 114 = Check temperature sensor 1 115 = Check temperature sensor 2 (if available) 118 = Check temperature sensor 3 (if available)
163	Not significant	"Don't care" bytes, can be any value

### Returned Array

Byte	Data	Description
0	114, 115 or 118	Interrupt code for Get Internal Temperature: 114 = Check temperature sensor 1 115 = Check temperature sensor 2 (if available) 118 = Check temperature sensor 3 (if available)
1	43 or 45	ASCII code for the first character of the temperature: 43 = positive (+) 45 = negative (-)
2	Temperature Digit 1	ASCII character code for the first digit of the temperature reading
3	Temperature Digit 2	ASCII character code for the second digit of the temperature reading
4	46	ASCII character code for the decimal point symbol (".")
5	Temperature Decimal Place 1	ASCII character code for the first decimal place of the temperature reading
6	Temperature Decimal Place 2	ASCII character code for the second decimal place of the temperature reading
7-63	Not significant	"Don't care" bytes, can be any value

### Example

To check the internal temperature measured by sensor 2, send the following transmit array:

Byte	Data	Description
0	115	Interrupt code for Get Internal Temperature @ Sensor 2

The below returned array would indicate a temperature of +28.43°C (see [Appendix A](#) for conversions between decimal, binary and ASCII characters):

Byte	Data	Description
0	115	Interrupt code for Get Internal Temperature @ Sensor 2
1	43	ASCII character code for +
2	50	ASCII character code for 2
3	56	ASCII character code for 8
4	46	ASCII character code for .
5	52	ASCII character code for 4
6	51	ASCII character code for 3

### See Also

SCPI: [Get Internal Temperature](#)

## 6 - Ethernet Control over IP Networks

Mini-Circuits' ZTM & RCM Series modular test systems have an RJ45 connector option for remote control over Ethernet TCP/IP networks. HTTP (Get/Post commands) and Telnet communication are supported. UDP transmission is also supported for discovering available ZTM Series systems on the network.

The system can be configured manually with a static IP address or automatically by the network using DHCP (Dynamic Host Control Protocol):

- Dynamic IP (factory default setting)
  - Subnet Mask, Network Gateway and local IP Address are assigned by the network server on each connection
  - The only user controllable parameters are:
    - TCP/IP Port (the port used for HTTP communication with the network; default is port 80)
    - Password (up to 20 characters; default is no password)
- Static IP
  - All parameters must be specified by the user:
    - IP Address (must be a legal and unique address on the local network)
    - Subnet Mask (subnet mask of the local network)
    - Network gateway (the IP address of the network gateway/router)
    - TCP/IP port (the port used for HTTP communication with the network; default is port 80)
    - Password (up to 20 characters; default is no password)

### Notes:

1. The TCP/IP port must be included in every HTTP command to the ZTM Series system unless the default port 80 is used
2. Port 23 is reserved for Telnet communication

## 6.1 - Ethernet Communication

Communication over Ethernet can be accomplished using HTTP Get/Post commands or Telnet communication to send the SCPI commands outlined in [SCPI Commands for Control of Modular Test Components](#). These communication protocols are both commonly supported and simple to implement in most programming languages. Any Internet browser can be used as a console/tester for HTTP control by typing the commands/queries directly into the address bar.

### 6.1 (a) - Sending SCPI Commands/Queries Using HTTP

The basic format of the HTTP command to send to the modular test system is:

<http://ADDRESS:PORT/PWD;COMMAND>

Where

- `http://` is required
- ADDRESS = IP address (required)
- PORT = TCP/IP port (can be omitted if port 80 is used)
- PWD = Password (can be omitted if password security is not enabled)
- COMMAND = Command to send to the switch

Example 1:

<http://192.168.100.100:800/PWD=123;;SPDT:1A:STATE:2>

Explanation:

- The ZTM Series has IP address 192.168.100.100 and uses port 800
- Password security is enabled and set to “123”
- The command is to set an SPDT in location 1A to state 2

Example 2:

<http://10.10.10.10/:SP4T:1:STATE?>

Explanation:

- The switch has IP address 10.10.10.10 and uses the default port 80
- Password security is disabled
- The command is to query the switch state of an SP4T in location 1

The system will return the result of the command/query as a string of ASCII characters.

## 6.1 (b) - Sending SCPI/Commands/Queries Using Telnet

Communication is started by creating a Telnet connection to the system's IP address. On successful connection the "line feed" character will be returned. If the system has a password enabled then this must be sent as the first command after connection.

The system can be optionally configured to return a full prompt to the user with each Telnet response. The prompt will take the form ZTM-X> where ZTM-X is the model name of the connected system. This feature can be enabled using the GUI application software or programmatically via the DLL.

The full list of all commands and queries is detailed in the following sections. A basic example of the Telnet communication structure using the Windows Telnet Client is summarized below:

- 1) Set up Telnet connection to a modular test system with IP address 10.0.6.46:



```

C:\WINDOWS\system32\telnet.exe
Welcome to Microsoft Telnet Client
Escape Character is 'CTRL+'
Microsoft Telnet> O 10.0.6.46
  
```

- 2) The "line feed" character is returned indicating the connection was successful:



```

Telnet 10.0.6.46
PWD=12345;
  
```


- 3) The password (if enabled) must be sent as the first command in the format "PWD=x;". A return value of "1 - Success" indicates success:



```

Telnet 10.0.6.46
PWD=12345;
1 - Success
  
```

- 4) Any number of commands and queries can be sent as needed:



```

Telnet 10.0.6.46
PWD=12345;
1 - Success
RUDAT:1A:STARTUPATT:INDICATOR?
F
RUDAT:1A:STARTUPATT:VALUE?
23.00
RUDAT:1A:MAX?
90.00
SP4T:1:SCOUNTER?
1;2;0;0
SPDT:1A:SCOUNTER?
0
MTS:1A:SCOUNTER?
0
LABEL:1A?
LABEL=VVVV XXXX
LABEL:1A:"PROJECT-AAA"
1 - Success
LABEL:1A?
LABEL="PROJECT-AAA"
  
```

## 6.1 (c) - Device Discovery Using UDP

In addition to HTTP and Telnet, ZTM & RCM Series test systems also provide limited support of the UDP protocol for the purpose of “device discovery.” This allows a user to request the IP address and configuration of all Mini-Circuits modular test systems connected on the network; full control of those units is then accomplished using HTTP or Telnet, as detailed previously.

Alternatively, the IP configuration can be identified or changed by connecting the system with the USB interface (see [Configuring Ethernet Settings](#)).

Note: UDP is a simple transmission protocol that provides no method for error correction or guarantee of receipt.

### UDP Ports

Mini-Circuits’ modular test systems are configured to listen on UDP port 4950 and answer on UDP port 4951. Communication on these ports must be allowed through the computer’s firewall in order to use UDP for device discovery. If the test system’s IP address is already known it is not necessary to use UDP.

### Transmission

The command **MODULAR-ZT?** should be broadcast to the local network using UDP protocol on port 4950.

### Receipt

All Mini-Circuits ZTM & RCM Series systems that receive the request will respond with the following information (each field separated by CrLf) on port 4951:

- Model Name
- Serial Number
- IP Address/Port
- Subnet Mask
- Network Gateway
- MAC Address

## Example

Sent Data:

**MODULAR-ZT?**

Received Data:

Model Name: ZTM-999  
Serial Number: 11302120001  
IP Address=192.168.9.101 Port: 80  
Subnet Mask=255.255.0.0  
Network Gateway=192.168.9.0  
Mac Address=D0-73-7F-82-D8-01

Model Name: ZTM-999  
Serial Number: 11302120002  
IP Address=192.168.9.102 Port: 80  
Subnet Mask=255.255.0.0  
Network Gateway=192.168.9.0  
Mac Address=D0-73-7F-82-D8-02

Model Name: ZTM-999  
Serial Number: 11302120003  
IP Address=192.168.9.103 Port: 80  
Subnet Mask=255.255.0.0  
Network Gateway=192.168.9.0  
Mac Address=D0-73-7F-82-D8-03

## 7 - Program Examples & Tutorials

These examples are intended to demonstrate the basics of programming with Mini-Circuits' ZTM & RCM Series test systems. If support is required for a specific programming example which isn't covered below then please contact Mini-Circuits for support ([testsolutions@minicircuits.com](mailto:testsolutions@minicircuits.com)).

### 7.1 - Perl Programming

#### 7.1 (a) - Ethernet HTTP Connection Using Perl's LWP Simple Interface

Perl's LWP Simple interface can be used to send HTTP commands to the ZTM or RCM Series when programming with Perl. The below code example demonstrates the process.

```
#!/usr/bin/perl
use strict;
use warnings;
use LWP::Simple;          # Use the LWP::Simple interface for HTTP

my $value = 40;
my $ip_address = "192.168.9.74";      # IP address of the ZTM Series to control

my $att_list = "1A,2A,3A,4A";        # The list of attenuator locations in the ZTM
my @att_location = split /,/, $att_list;

foreach my $att_location (@att_location) {
    # Loop for each attenuator location

    # Set attenuator in this location
    my $return_value = get("http://$ip_address/:RUDAT:$att_location:ATT:$value");
    print "ZTM Series response: $return_value\n";

    # Confirm attenuation setting for this attenuator
    $return_value = get("http://$ip_address/:RUDAT:$att_location:ATT?");
    print "Attenuator $att_location set to $return_value\n";
}
```

#### 7.1 (b) - USB Connection Using the ActiveX DLL in 32-bit Perl Distributions

The majority of 32-bit Perl distributions for Windows operating systems provide support for ActiveX, meaning Mini-Circuits' ActiveX DLL can be used to control the modular test system in these environments. The below simple code segment demonstrates the process for this; any number of commands can be sent between the Connect and Disconnect functions.

```
use feature ':5.10';
use Win32::OLE;
use Win32::OLE::Const 'Microsoft ActiveX Data Objects';

my $ztm = Win32::OLE->new('ModularZT.USB_Control');

$ztm ->Connect();
$ztm ->Send_SCPI(":RUDAT:1A:ATT:70.25", $ztm_return)
$ztm ->Send_SCPI(":SP4T:1A:STATE:3", $ztm_return)
$ztm ->Disconnect;
```



## 7.1 (c) - Work-Around for 64-bit Perl Distributions Using USB Connection

The majority of 64-bit Perl distributions do not provide support for either ActiveX or .Net so in these cases Mini-Circuits' DLLs cannot be used directly. The work-around when a USB connection is required is to create a separate executable program in another programming environment which can sit in the middle. The function of the executable is to use the .Net DLL to connect to the ZTM or RCM Series, send a single user specified command, return the response to the user, and disconnect from the DLL. This executable can then be easily called from Perl script to send the required commands to the system, without Perl having to directly interface with the DLL.

Mini-Circuits can supply on request an executable to interface with the DLL. See [Creating an Executable Using the .Net DLL in C# for USB Control](#) for the example source code for such an executable (developed using C#). The below script demonstrates use of this executable in Perl script to send a SCPI command to a ZTM Series test system (specified by serial number or address) and read the response.

```
#!/usr/bin/perl
use strict;
use warnings;

my $serial_number = 11404280010;      # The ZTM Series serial number
my $att_list = "1A,2A,3A,4A";        # The list of attenuator locations in the ZTM
my @att_location = split /,/, $att_list;
my $value = 40;

my $exe = "ZTM.exe";                 # The .exe providing an interface to the ZTM DLL
my @cmd;

foreach my $att_location (@att_location) {

    # Loop for each attenuator location

    # Set attenuator in this location
    @cmd = ($exe, "-s $serial_number :RUDAT:$att_location:ATT:$value");
    my $return_value = qx{@cmd};
    print "ZTM Series response: $return_value\n";

    # Confirm attenuation setting for this attenuator
    @cmd = ($exe, "-s $serial_number :RUDAT:$att_location:ATT?");
    $return_value = qx{@cmd};
    print "Attenuator $att_location set to $return_value\n";

}
```

## 7.2 - C# Programming

### 7.2 (a) - Creating an Executable Using the .Net DLL in C# for USB Control

The below example is a simple executable program that connects to the .Net DLL, sends a user specified SCPI command to the test system, returns the response, then disconnects from the DLL and terminates. It requires the .Net DLL to be installed on the host operating system and the ZTM or RCM Series test system to be connected to the PC via USB.

```
namespace ZTM
{
    class Program
    {
        static int Main(string[] args)
        {
            int x = 0;
            string SN = null;
            string SCPI = null;
            string RESULT = null;
            int Add = 0;
            ModularZT64.USB_ZT ZT;           // Reference the DLL
            if (args.Length == 0) return 0;
            ZT = new ModularZT64.USB_ZT (); // Declare a class (defined in the DLL)
            SCPI = args[2];
            if (args[0].ToString().Contains("-help")) // Print a help file
            {
                Console.WriteLine("Help ZTM.exe");
                Console.WriteLine("-----");
                Console.WriteLine("ZTM.exe -s SN command :Send SCPI command to S/N");
                Console.WriteLine("ZTM.exe -a add SCPI :Send SCPI command to Address");
                Console.WriteLine("-----");
            }
            if (args[0].ToString().Contains("-s")) // User want to connect by S/N
            {
                SN = args[1];
                x = ZT.Connect(ref SN);           // Call DLL connect function
                x = ZT.Send_SCPI(ref SCPI, ref RESULT); // Send SCPI command
                Console.WriteLine(RESULT);        // Return the result
            }
            if (args[0].ToString().Contains("-a")) // User wants to connect by address
            {
                Add = Int16.Parse(args[1]);
                x = ZT.ConnectByAddress(ref Add);
                x = ZT.Send_SCPI(ref SCPI, ref RESULT);
                Console.WriteLine(RESULT);
            }
            ZT.Disconnect(); // Call DLL disconnect function to finish
            return x;
        }
    }
}
```

This executable can be called from a command line prompt or within a script. The following command line calls demonstrate use of the executable (compiled as ZTM.exe), connecting by serial number or address, to set and read attenuation:

- `ZTM.exe -s 11401250027 :RUDAT:1A:ATT:35.75` (serial number 11401250027)
- `ZTM.exe -a 255 :RUDAT:1A:ATT?` (USB address 255)

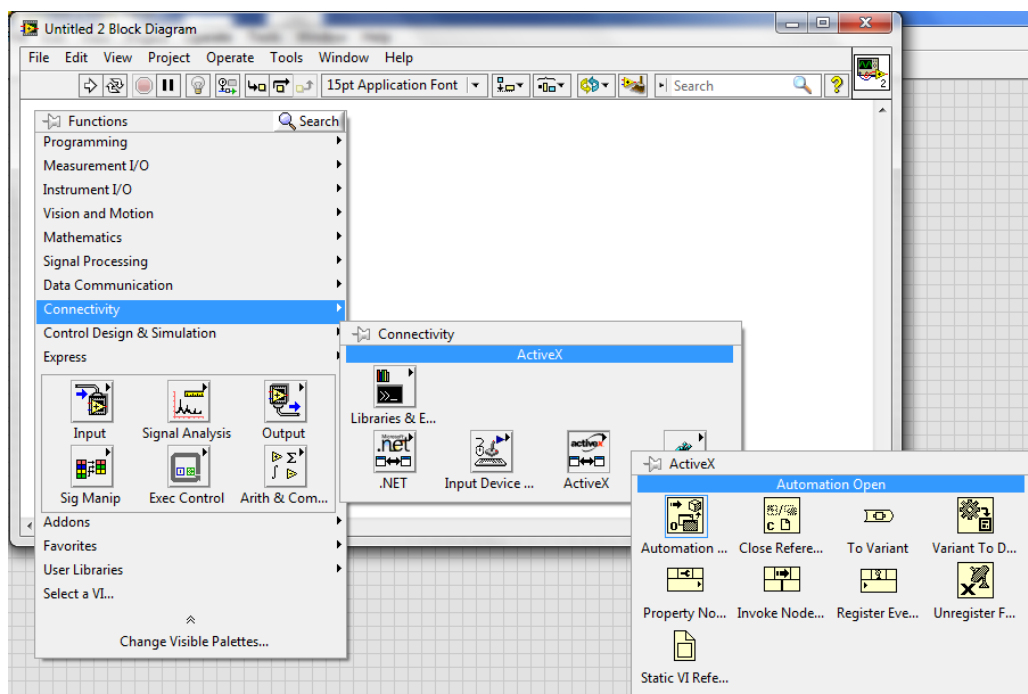
## 7.3 - LabVIEW

### 7.3 (a) - Creating a LabVIEW VI for USB Control with the ActiveX DLL

These instructions demonstrate how to set up a LabVIEW VI for control of Mini-Circuits' ZTM and RCM Series test systems using the ActiveX DLL file when connected by USB.

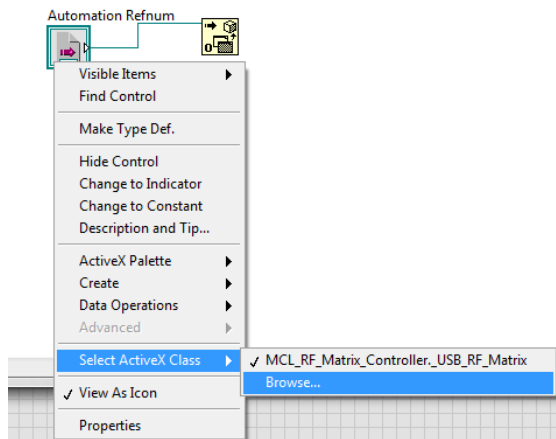
#### 1. Create a New VI (Virtual Instrument) and Reference the DLL

- a. Open LabVIEW with a new, blank VI.
- b. In the Block Diagram window (accessible by pressing Ctrl+E), select Functions Palette from the “View” menu at the top of the screen.
- c. Click through the Connectivity palette to the ActiveX sub-palette. Select the **Automation Open** function and place it on the block diagram.



- d. Right click on the *Automation Refnum* terminal on the left of the **Automation Open** function and create a control.

- e. Right click on the new control, choose the 'Select ActiveX Class' option and browse to the location of the ModularZT.dll file.

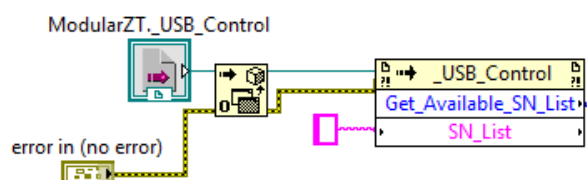


- f. After selecting the DLL file, choose "USB\_Control" from the list of objects presented.
- g. Right click on the *Error In* terminal of the **Automation Open** function and create a new control.
- h. To save space on the block diagram, right-click the **Error In** icon and uncheck the "View As Icon" option.

## 2. Identifying the Serial Numbers of all Connected ZTM Series Systems

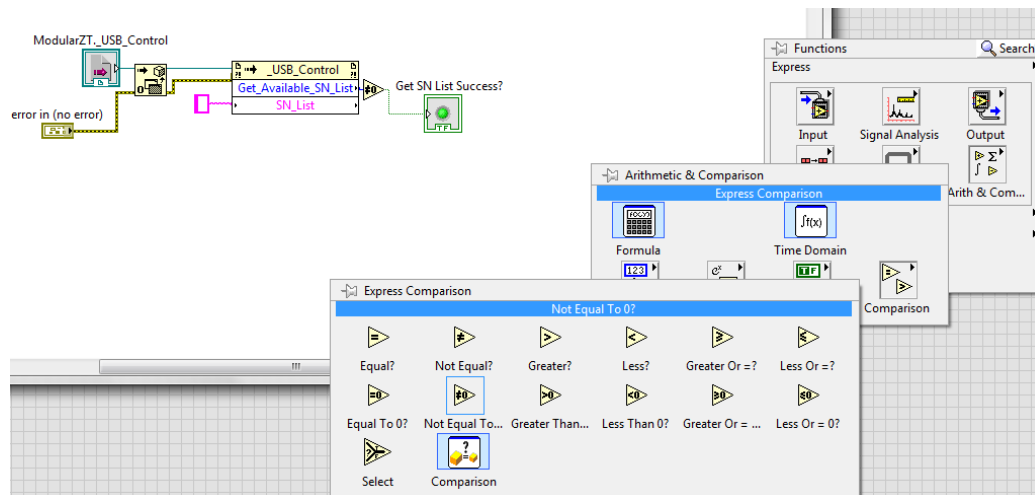
This section makes use of the Get\_Available\_SN\_List DLL function to provide a drop-down list of all connected serial numbers, allowing the user to choose which system to connect. If the serial numbers are already known or only a single system is connected then this process can be omitted.

- a. From the ActiveX sub-palette, choose **Invoke Node** and place the node on the block diagram.
- b. Connect the right *Automation Refnum* terminal of the **Automation Open** function to the *Reference* terminal on the **Invoke Node**.
- c. Connect the *Error Out* terminal of the **Automation Open** function to the *Error In* terminal of the **Invoke Node**.
- d. Click on the Method of the **Invoke Node** to display a list of all available functions (defined in the ModularZT.dll file), select "Get\_Available\_SN\_List".
- e. Right-click the *Input* terminal of the SN\_List parameter and create a blank constant. The constant will provide the SN\_List parameter to the Get\_Available\_SN\_List function.

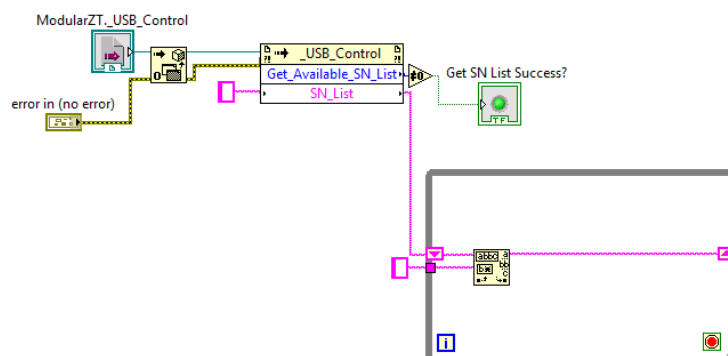


- f. From the Programming palette, go to the Comparison sub-palette. Select the **Not Equal To 0** function and place it on the block diagram.
- g. Connect the *Get\_Available\_SN\_List* terminal of the **Invoke Node** to the input of the **Not Equal To 0** function.

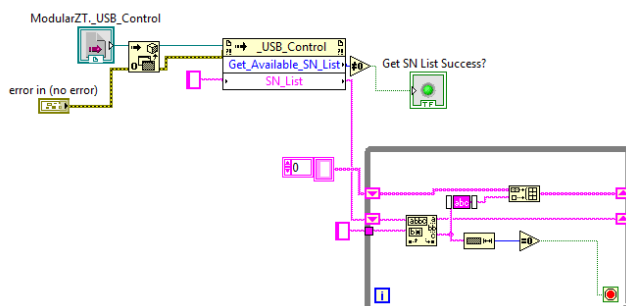
- h. Create an **Indicator** at the *Output* terminal of the **Not Equal To 0** function. The output of the *Get\_Available\_SN\_List* function will be 0 (failure to connect) or 1 (successfully connected) so the indicator should light up when the retrieval of the serial numbers is successful.
- i. Right-click on the **Indicator** and rename to “Get SN List Success?”.



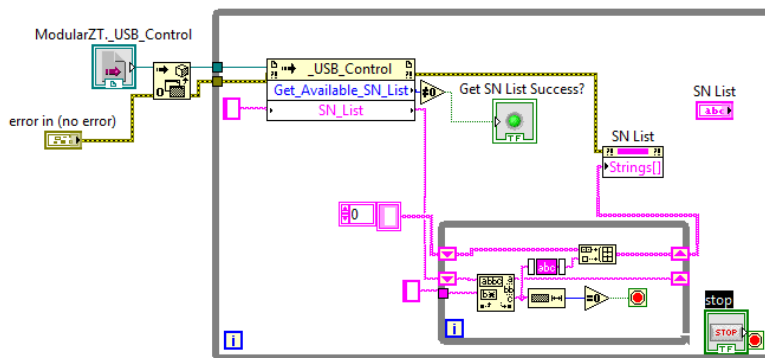
- j. Place a **While Loop** (found in the Programming palette, Structures sub-palette) on the block diagram, external to the previous objects.
- k. Delete the stop button if it was included automatically but not the loop condition (the red dot).
- l. Place a **Match Pattern** function (found in the Programming palette, String sub-palette) inside the **While Loop**.
- m. Connect the output of the “SN\_List” parameter (from the *Get\_Available\_SN\_List* node) to the *String* input terminal of the **Match Pattern** function.
- n. Create another empty string constant outside the **While Loop**, connected to the *Regular Expression* terminal of the **Match Pattern** function. This is because each SN that the Node outputs will be separated by a blank space.
- o. Connect the *Before Substring* terminal of the **Match Pattern** function to the right-hand edge of the **While Loop**.
- p. Right-click on the **Loop Tunnel** between the “SN\_List” parameter and the *String* input terminal of the **Match Pattern**, select ‘Replace with Shift Register’. The mouse cursor will automatically change to signify the other end of the **Shift Register**, click on the **Loop Tunnel** at the right-hand edge of the **While Loop** to place it.



- q. Place a **Trim Whitespace.vi** from the String sub-palette inside the **While Loop**.
- r. Connect the *After Substring* terminal of the **Match Pattern** function to the input of the **VI**.
- s. Place a **Build Array** function (found in the Programming palette, Array sub-palette) in the **While Loop** and expand it to have two inputs by dragging the bottom edge down
- t. An empty **String Array** constant is needed for the **Build Array** function. Select **Array** constant from the Array sub-palette and place it outside the **While Loop**.
- u. Place another **String** constant on the block diagram and drag it into the **Array** constant box to create the empty **String Array**.
- v. Connect the **String Array** constant to the first *Input* terminal of the **Build Array** function.
- w. Connect the *Trimmed String* terminal of the **Trim Whitespace** VI to the second terminal of the **Build Array** function.
- x. Connect the output of the **Build Array** function to the edge of the **While Loop** and create another **Shift Register**, with the other end at the empty **String Array Loop Tunnel**.
- y. Place a **String Length** function (from the String sub-palette) inside the **While Loop**.
- z. Connect the *Input* terminal of the function to the *After Substring* terminal of the **Match Pattern** function. This will form a junction since the *After Substring* terminal is also connected to the **Trim Whitespace** VI.
- aa. Connect the *Length* output terminal of the **String** Length function to the input of a new **Equal to 0** function (found in the Comparison sub-palette).
- bb. Connect the output of the **Equal to 0** function to the loop condition. If the output of the **String Length** function is 0 it will indicate that there are no more serial numbers and the **Equal to 0** operator will cause the loop to stop.



- cc. On the Front Panel of this VI, create a drop-down menu by placing a **System Combo Box** function (found in the Systems palette, String & Path sub-palette).
- dd. On the Block Diagram, right click the corresponding **System Combo Box** function, create a **Strings[] Property Node** by selection "Create", "Property Node", then "Strings[]". Place the **Strings[] Property Node** outside the **While Loop**.
- ee. Right click the **Strings[] Property Node** and select "Change to Write".
- ff. Rename the **System Combo Box** function to "SN\_List". Right-click and un-tick "View As Icon" to save space on the block diagram.
- gg. Connect the output from the **Shift Register** that follows the **Build Array** function to the input of the **Strings[] Property Node**.
- hh. Connect the *Error Out* terminal of the **\_USB\_Control Node** to the *Error In* terminal of the **Strings[] Node**.
- ii. Create another **While Loop** and arrange it so that it encompasses everything from the **Automation Open** function onwards.
- jj. If a **Stop Button** was not created automatically then right-click on the loop condition and select "Create Control" to place the button in the loop.



kk. On the Front Panel, change the **Stop Button** text from “Stop” to “Connect”.

### 3. Connecting to a ZTM Series Test System

- Create a new **Invoke Node** outside the **While Loop**.
- Connect the *Reference Out* terminal of the **Get\_Available\_SN\_List** Node to the *Reference* terminal of the new **Invoke Node**.
- Select “Connect” as the method for the new node.
- Connect the *Error Out* terminal of the **Strings[]** node to the *Error In* terminal of the **Connect** node.
- Connect the output terminal of the SN\_List combo box to the SN input terminal of the **Connect** Node.
- During execution, the program will not get to this stage until the **While Loop** has exited, the result being that the program will populate the drop-down box with all serial numbers and wait for a user input. The program will continue when the user selects the desired serial number and clicks the **Connect** button. If the process of identifying serial numbers is not required (see step 3 above) then the “Connect” function can be used in place of the “Get\_Available\_SN\_List” function and everything that followed.
- Following the **Connect** Node, the user can place any number of additional nodes in sequence to perform all operations required of the ZTM-X system. The *Reference In* and *Error In* terminals of each new node should be connected to the *Reference Out* and *Error Out* terminals of the previous nodes.
- The final **Invoke Node** in the program should be set with the “Disconnect” function in order to properly close the connection to the ZTM-X system.
- The final step in the LabVIEW sequence is to create a **Close Reference** function from the Connectivity palette to terminate the reference to the DLL file. The *Reference In* and *Error In* terminals of the **Close Reference** function should be connected to the respective terminals of the **Disconnect** function.
- An **Error Out** indicator should be added by right clicking on the *Error Out* terminal of the **Close Reference** function and creating an indicator to show the result.

